UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

YURI S. CAMARGO
HERBERT S. SANTOS

LogicAPI: Web Service para processamento de consequências lógicas

RIO DE JANEIRO
2018

YURI S. CAMARGO
HERBERT S. SANTOS

LogicAPI: Web Service para processamento de consequências lógicas

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Mário Roberto Folhadela Benevides

RIO DE JANEIRO
2018

Camargo, Yuri Santana

C172l LogicAPI: processamento de fórmulas lógicas aplicadas ao ensino de Lógica / Yuri Santana Camargo, Herbret Salazar dos Santos. – Rio de Janeiro, 2018.

91 f.

Orientador: Mário Roberto Folhadela Benevides.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal do Rio de Janeiro, Instituto de Matemática, Bacharel em Ciência da Computação, 2018.

1. Lógica. 2. Webservice. 3. API. I. Santos, Herbert Salazar dos. II. Benevides, Mário Roberto Folhadela. III. Universidade Federal do Rio de Janeiro, Instituto de Matemática (Orient). IV. Título.

YURI S. CAMARGO
HERBERT S. SANTOS

LogicAPI: Web Service para processamento de consequências lógicas

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em ___ de _____ de _____

BANCA EXAMINADORA:

Mário Roberto Folhadela Benevides
Ph.D. (Imperial College)

Gerson Zaverucha
Ph.D. (Imperial College)

Geraldo Bonorino Xexéo
D.Sc (Coppe - UFRJ)

AGRADECIMENTOS

Yuri Santana Camargo

Quero agradecer primeiramente a Deus, por permitir que eu ingressasse numa universidade de excelência e permitisse que caminhasse até o final.

Agradeço aos professores que fizeram parte da minha vida na graduação, adquiri experiências muito boas que contribuíram tanto pessoalmente quando profissionalmente nesta jornada.

Dedico este trabalho bem como todo o resultado já conquistado com a minha graduação aos meus pais Maria e Julio que desde minha infância apesar das dificuldades financeiras me apoiaram o máximo possível para que eu estudasse em escola particular e posteriormente tivesse uma base boa para seguir o meu caminho e chegar aonde cheguei.

Agradeço também ao Herbert e ao Marco por toda a contribuição que tiveram nesse trabalho. Herbert apesar das dificuldades com o trabalho e posteriormente o mestrado manteve o foco para conseguirmos a realização do projeto. Marco, apesar das dificuldades com relação as decisões que foram necessárias na graduação, mesmo não participando mais do projeto continuou com o seu apoio o que nos permitiu concluir o projeto.

AGRADECIMENTOS

Herbert Salazar dos Santos

"Como agradecer por tudo que fizeste? Não merecedor, mas provaste o Seu amor sem fim! As vozes de um milhão de anjos não expressam a minha gratidão. Tudo o que sou e o que almejo ser eu devo tudo a Ti. A Deus seja a glória por tudo o que fez por mim. Quero viver para Ti, Tua vontade obedecer, e se o aplauso eu receber no calvário irei me gloriar. Com Seu sangue lavou-me, Seu poder levantou-me! A Deus seja a glória para sempre, amém!".

Faço minhas as palavras de Victorino Silva na música "Meu Tributo". Tudo o que consegui até hoje foi porque o Senhor me sustentou e me guiou, mesmo sem eu merecer nada e sendo mais vil pecador. Agradeço também por ter me dado pais tão maravilhosos que me ajudaram em toda a minha caminhada. Sou grato ainda por todas as oportunidades que tive dentro da Universidade Federal do Rio de Janeiro, que contribuíram para que eu conseguisse me formar.

Não poderia deixar de citar também os grandes amigos que fiz na minha caminhada escolar até chegar a este ponto, que são os melhores amigos que alguém pode ter. A animação que eles proporcionaram em todos esses anos me ajudou a superar os obstáculos que enfrentei. A todos vocês, muito obrigado!

"Nada é suficientemente bom.
Então vamos fazer o que é certo,
dedicar o melhor de nossos esforços para atingir o inatingível,
desenvolver ao máximo os dons que Deus nos concedeu
e nunca parar de aprender."

Beethoven

RESUMO

O presente trabalho apresenta uma ferramenta capaz de determinar consequências lógicas através de diferentes métodos, focando em métodos ensinados na disciplina de Lógica de Ciência da Computação da Universidade Federal do Rio de Janeiro. Também é apresentada uma aplicação de usuário que se utiliza dos serviços providos pela ferramenta para auxiliar o ensino da disciplina de lógica ministrada pelo professor Mario Roberto Folhadelha Benevides. Inicialmente são discutidos conceitos de Lógica para o entendimento da utilização da solução bem como as terminologias e o escopo coberto neste trabalho. Em seguida é apresentando uma panorâmica sobre a evolução tecnológica, a proposta de solução e efetivamente sua implementação.

Palavras-chave: lógica. *webservice*. API.

ABSTRACT

The present work presents a tool capable of determining logical consequences through different methods, focusing on methods taught in the discipline of Computer Science Logic of the Federal University of Rio de Janeiro. Also an user application that uses the services provided by the tool to help the teaching of the discipline of logic taught by professor Mario Roberto Folhadelha Benevides was built. Initially, concepts of Logic are discussed for the understanding of the use of the solution as well as the terminologies and the scope covered in this work. Then an overview of the technological evolution, the proposed solution and effectively its implementation is showed.

Keywords: logic. webservice. API.

LISTA DE ILUSTRAÇÕES

LISTA DE ALGORITMOS

SUMÁRIO

1 INTRODUÇÃO

Na sociedade contemporânea, com o advento da rápida evolução tecnológica, uma grande parcela da população tem tido fácil acesso aos recursos de computação e *web* (TRACTO, 2017) (GROUP, 2018a). A quantidade de usuários que utilizam a internet mais do que dobrou em 8 anos (TRACTO, 2017), chegando a praticamente metade da população mundial com acesso a *sites* e sistemas *web* (GROUP, 2018a). Ao levarmos em consideração o Brasil, atualmente mais de 60% da população é usuária da rede mundial de computadores (GROUP, 2018b). Ao mesmo passo, o número de *sites* e sistemas *web* tem crescido assustadoramente, passando de apenas 1 em 1991 para 1 bilhão em 2014 e chegando próximo de 2 bilhões em 2018 (STATS, 2018).

Portanto não é difícil observar que existe hoje em todas as áreas de estudo as mais variadas ferramentas *online* para auxiliar na resolução ou até mesmo resolver toda sorte de problemas utilizando diversos métodos diferentes (DUMON, 2013). A área de lógica não é diferente. Na seção 1.1 são expostas algumas ferramentas criadas por pesquisadores a fim de resolverem alguns problemas lógicos específicos. Entretanto tais ferramentas só permitem a sua utilização da forma como foram concebidas, não sendo possível integrá-las com outros sistemas para que juntos possam se adequar a problemas distintos de natureza similar.

O objetivo deste trabalho é, portanto, apresentar a LogicAPI: uma ferramenta desenvolvida para ser capaz de determinar consequências lógicas através de diferentes métodos encontrados na literatura, e que atenda a qualquer tipo de aplicação que necessite da verificação e resolução de problemas desta natureza, oferecendo diversos serviços construídos segundo o conceito de *Web Services*.

Utilizaremos, ainda, a LogicAPI para solucionar um problema real desta natureza, construindo uma aplicação que seja capaz de sanar a necessidade observada pelo professor Mario Roberto Folhadela Benevides ao lecionar a disciplina de lógica para o curso de Ciência da Computação da Universidade Federal do Rio de Janeiro, de possuir alguma ferramenta que auxiliasse os alunos durante o curso. Tal aplicação se valerá dos serviços implementados pela LogicAPI e funcionará como um sistema de auxílio ao aprendizado da disciplina, utilizando os dados processados pelo *Web Service* e os apresentando de forma amigável para facilitar o entendimento. Com a proposta de uma interface fácil e intuitiva, a aplicação ajudará os alunos de maneira completa para que possam otimizar o tempo de aprendizado, levando a uma melhora significativa no desempenho dos alunos. A figura 1 mostra a página inicial da aplicação.

A LogicAPI, por se utilizar do conceito de *Web Service* e permitir posterior integração a outros sistemas que possam ser gerados, possui a vantagem de ser facilmente expandida e renovada, adequando-se às tecnologias atuais e a novas tecnologias que forem surgindo.

Figura 1 – Tela Inicial da aplicação de auxílio à disciplina de lógica

O trabalho está organizado da seguinte forma: a seguir serão listados trabalhos relacionados à área de lógica que possuem ferramentas *online*. No capítulo 2 serão apresentados os conceitos de lógica necessários ao entendimento do problema e solução de consequências lógicas. No capítulo 3 serão apresentados os conceitos e as tecnologias utilizadas para construção da LogicAPI, bem como a justificativa da escolha por cada uma delas. No capítulo 4 entraremos em detalhes da implementação do *Web Service* e seus diferentes serviços. Por fim, no capítulo 5, exemplificaremos o funcionamento da aplicação desenvolvida para auxílio no curso de lógica lecionado pelo professor Mario Roberto Folhadela Benevides.

1.1 TRABALHOS RELACIONADOS

Os trabalhos mencionados neste texto possuem abordagens e propostas diferentes mas um objetivo em comum que é o de auxiliar alunos nos aprendizados de disciplinas em universidades, ou mesmo entusiastas que estejam adquirindo algum conhecimento por conta própria. O grande diferencial do nosso trabalho em relação aos trabalhos apresentados é que a nossa ferramenta é uma API de Web Service que processa fórmulas lógicas e não depende de um front-end específico como os trabalhos abaixo. Assim qualquer aplicação para propósitos mais gerais pode ser produzida a partir da nossa ferramenta. No caso do trabalho em apresentado neste texto, produziu-se como resultado da API uma aplicação que soluciona exercícios de lógica.

1.1.1 Data Structures Visualization

O professor associado David Galles da Universidade de São Francisco (USFC) desenvolveu uma ferramenta online para a visualização do funcionamento de algoritmos no contexto de estrutura de dados. Na visão do professor o melhor jeito de aprender complexas estruturas de dados é observá-las em ação. Tal trabalho contribuiu na inspiração do trabalho descrito nesta monografia por compartilharmos da mesma visão deste professor. A ferramenta funciona similar a proposta dessa solução, no que diz respeito a solução dinâmica no browser, apesar do contexto ser diferente. Este trabalho pode ser visto em https://www.cs.usfca.edu/ galles/visualization/Algorithms.html

1.1.2 Logic Tools

Consiste em uma aplicação para validação de fórmulas lógicas. Dada uma expressão lógica qualquer, esta ferramenta aplica alguns métodos como resolução e tabela verdade para validar uma expressão lógica. A ferramenta é funcional, porém não mostra as etapas do cálculo. A aplicação possui código aberto e pode ser vista em http://logictools.org/about.html

1.1.3 The Propositional Logic Calculator

O professor Enrico Francini da Free University of Bozen-Bolzano desenvolveu esta aplicação para auxiliar no aprendizado de lógica. A aplicação é relativamente limitada, pois apenas 3 átomos podem ser inseridos. Estes átomos são usados para compôr fórmulas lógicas e é feito um cálculo em todas as possibilidades de valoração lógica para cada átomo como uma tabela verdade. A aplicação pode ser vista em http://www.inf.unibz.it/ franconi/teaching/propcalc/

1.1.4 Natural deduction proof editor and checker

Esta aplicação consiste num verificador de Deduções naturais. Ela permite que o usuário insira um problema que se queira provar via dedução e cada passo da dedução pode ser inserido manualmente, fornecendo para o usuário uma verificação se o passo pode ser dado ou não. É bastante útil para propósitos tanto de estudo quanto gerais e ajudou nosso trabalho na elaboração da nossa dedução natural. Segue o link abaixo

http://proofs.openlogicproject.org

1.1.5 App Web Natural Deduction

Esta aplicação consiste num solucionador de deduções naturais. Dada uma entrada do usuário, este solucionador realiza passos afim de se chegar a conclusão, porém a interface não é intuitiva e não há documentação, apenas exemplos para se guiar. O que nos forneceu

a informação de como tornar a interface intuitiva e informativa para o usuário da aplicação resultante do nosso trabalho. Segue o link abaixo:

http://teachinglogic.liglab.fr/DN/index.php

1.1.6 App Windows Natural Deduction

Consiste em uma aplicação desktop para Windows bem completa que permite que o usuário em uma interface orientada e com bastante liberdade resolva suas deduções. As deduções são resolvidas automaticamente e o usuário só precisa entrar com sua proposição lógica. Segue o link abaixo:

https://www.microsoft.com/en-us/p/naturaldeduction/9ndb7hz5pfm0

1.1.7 Tree Proof Generator

Consiste em um solucionador de Tableaux que ocorre a partir de uma proposição lógica dada como entrada. Ele gera como resultado uma árvore via animação. Segue o link abaixo:

https://www.umsu.de/logik/trees/

2 FUNDAMENTAÇÃO TEÓRICA

Para se entender como o sistema funciona, bem como seus resultados, é importante se ter em mente os conceitos teóricos básicos da disciplina de lógica e este capítulo se destina a mostrar tais conceitos, bem como os métodos ensinados na disciplina que foram implementados no nosso sistema.

2.1 O QUE É LÓGICA

De acordo com (BENEVIDES, 2008), Lógica é o estudo do raciocínio dedutivo, explicando de maneira informal, ou um sistema formal, explicando de maneira formal. O estudo da Lógica pode ser dividido em 3 partes fundamentais, sendo elas: **Linguagem**, **Regras de Dedução** e **Semântica**. A **linguagem** é utilizada para descrever o conhecimento que se deseja representar. As **Regras de Dedução** são utilizadas para se tirar conclusões a partir do conhecimento representado na linguagem. Por fim, a **Semântica**, é utilizada para dar significado aos objetos descritos na linguagem. No curso de lógica , são mostradas duas linguagens fundamentais e métodos que sobre elas operam para se tirar conclusões a partir das regras de dedução, bem como a semântica para expressar significado a seus objetos, os métodos e a semântica podem ser enxergados na mesma hierarquia, pois podem ser tiradas conclusões dependendo de como se *interpreta* a semântica num dado contexto. As linguagens que abordaremos a seguir são a **Linguagem da Lógica Clássica Proposicional** e a **Linguagem da Lógica Clássica de Primeira Ordem**.

2.2 LINGUAGEM DA LÓGICA CLÁSSICA PROPOSICIONAL

2.2.1 Linguagem

A linguagem proposicional é uma linguagem formal que possui como objetivo representar trechos de discurso de uma maneira precisa e sem ambiguidades. Quando se apresenta uma linguagem formal, é necessário inicialmente fornecer os componentes básicos da linguagem, chamados de *alfabeto*, para então fornecer as *regras de formação* da linguagem, também chamadas de *gramática*. Na lógica proposicional, o alfabeto é composto pelos seguintes elementos:

- Um conjunto infinito e contável de *símbolos proposicionais*, também chamados de *átomos*, ou de *variáveis proposicionais*. Para padronização da notação usaremos o conjunto das letras maiúsculas: $P = \{A, B, ...\}$.

- *Conectivos proposicionais*: $\neg$ (negação,lê-se: NÃO), $\vee$ (conjunção,lê-se: E), $\wedge$(disjunção, lê-se:OU), $\rightarrow$(implicação, lê-se: SE...ENTÃO...).
- *Símbolos de pontuação*: $(,)$;
- *Símbolos de verdade*: $verdadeiro, falso$;

Agora será apresentada a gramática por meio da qual serão definidas quais são as fórmulas bem formadas da linguagem. A noção de fórmula bem formada, ou simplesmente fórmula, é definida, indutivamente, pelas seguintes condições:

- Qualquer símbolo proposicional é uma fórmula.
- Se α e β são fórmulas então $(\alpha \wedge \beta)$,$(\alpha \vee \beta)$,$\neg\alpha$,$(\alpha \rightarrow \beta)$ também o são;
- Nada mais é fórmula.

De maneira alternativa a gramática da linguagem proposicional pode ser expressa por meio da notação *BNF*.

$$\alpha ::= P \mid (\alpha_1 \wedge \alpha_2) \mid (\alpha_1 \vee \alpha_2) \mid (\alpha_1 \rightarrow \alpha_2) \mid \neg\alpha$$

Convém também salientar que algumas vezes o símbolo $\leftrightarrow$ (lê-se: SE E SOMENTE SE) pode aparecer. Este conectivo é definido como:

$$(\alpha \leftrightarrow \beta) \equiv ((\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha))$$

2.2.2 Semântica

A semântica na lógica proposicional consiste na atribuição de significado às fórmulas da Linguagem. Esse significado se dá pelos *valores verdade* de modo que cada fórmula pode possuir como valor verdade V para verdadeiro ou F para falso. O valor verdade resultante de uma fórmula depende unicamente dos valores atribuídos aos seus símbolos proposicionais. Para simplificar, será mostrada a tabela verdade que cada conectivo pode gerar dependendo dos valores dos símbolos que compõe a fórmula. Seguem abaixo as tabelas:

Conjunção		
A	B	$A \wedge B$
V	V	V
V	F	F
F	V	F
F	F	F

Disjunção (Não-exclusiva)		
A	B	$A \wedge B$
V	V	V
V	F	V
F	V	V
F	F	F

Conjunção	
A	$\neg A$
V	F
F	V

Implicação		
A	B	$A \wedge B$
V	V	V
V	F	F
F	V	V
F	F	V

2.2.3 Consequências Lógicas Proposicionais

Dada toda discussão agora pode-se discorrer sobre o problema central que o curso de lógica foca em resolver e por sua vez o que será de fato implementado no sistema.

Definição 2.2.1. Seja α uma fórmula e Γ um conjunto de fórmulas:

1. Uma atribuição de valor verdade $\mathbf{v}$:$P \mapsto \{V, F\}$ satisfaz α se e somente se $v(\alpha) = V$. E $\mathbf{v}$ satisfaz Γ se e somente se v satisfaz cada membro de Γ.

2. Γ é satisfatível se e somente se existe uma atribuição $\mathbf{v}$ que satisfaz Γ. Caso contrário, Γ é insatisfatível.

Definição 2.2.2. Uma fórmula β é dita uma **consequência lógica** de um conjunto de fórmulas $\Gamma = \{\alpha_1, \alpha_2, ..., \alpha_n\}$, ou β é uma **implicação lógica** de $\Gamma, \alpha_1, \alpha_2, ..., \alpha_n \models \beta$, se somente se para toda valoração $\mathbf{v}$ se $v(\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_n)$

, então $v() = V$ Deste modo o objetivo é que dado um banco de dados (conjunto de fórmulas $BD = \{\alpha_1, ..., \alpha_n\}$ e uma pergunta (fórmula), α como saber se o banco de dados implica logicamente na pergunta? Os métodos que serão apresentados a seguir resolvem este problema.

2.2.4 Método de Dedução Natural

A dedução natural é um método que se utiliza de somente de regras de inferência, de modo que para cada conectivo lógico existem duas regras são elas: A **Regra de Introdução**: na qual uma fórmula contendo o conectivo pode ser inferida e a **Regra de Eliminação** na qual podem-se tirar consequências de uma fórmula contendo o conectivo.

Seguem abaixo as regras de inferência da dedução natural:

Conjunção $\wedge$

$\wedge - \mathbf{I}$

$$\frac{\alpha \quad \beta}{\alpha \wedge \beta}$$

$\wedge - \mathbf{E}$

$$\frac{\alpha \wedge \beta}{\alpha} \qquad \frac{\alpha \wedge \beta}{\beta}$$

Disjunção $\vee$

$\vee - \mathbf{I}$

$$\frac{\alpha}{\alpha \vee \beta} \qquad \frac{\beta}{\alpha \vee \beta}$$

$\vee - \mathbf{E}$

Onde $[\alpha]^i$ e $[\beta]^{i+1}$ são suposições de nível i e $i+1$ na dedução natural, respectivamente.

$$\frac{\alpha \vee \beta \qquad \begin{array}{c}[\alpha]^i \\ \vdots \\ \gamma\end{array} \qquad \begin{array}{c}[\beta]^{i+1} \\ \vdots \\ \gamma\end{array}}{\gamma}$$

Implicação $\rightarrow$

$\rightarrow - \mathbf{I}$

Onde $[\alpha]^i$ é uma suposição de nível i na dedução natural

$$\frac{\begin{array}{c}[\alpha]^i \\ \vdots \\ \beta\end{array}}{\alpha \rightarrow \beta^i}$$

$\rightarrow - \mathbf{E}$

$$\frac{\alpha \quad \alpha \rightarrow \beta}{\beta}$$

Negação $\neg$

$\neg - \mathbf{I}$

$$\frac{\alpha}{\neg\neg\alpha}$$

$\neg - \mathbf{E}$

$$\frac{\neg\neg\alpha}{\alpha}$$

Redução ao Absurdo $\rightarrow$
$\neg$ **– RAA**

$$\dfrac{\begin{array}{c}[\neg\alpha]\\ \vdots \\ \bot\end{array}}{\alpha}$$

$\neg$ **– ABS**

$$\dfrac{\alpha \wedge \neg\alpha}{\beta}$$

Dado que foram apresentadas as regras da dedução natural, convém agora formalizar a definição e apresentar alguns exemplos.

Definição 2.2.4.1. :

(i) Uma **prova** em dedução natural de uma fórmula φ a partir de um conjunto de fórmulas $BD = \{\varphi 1, \varphi 2, ..., \varphi k\}$ é uma sequência de fórmulas rotuladas da seguinte forma:

1. As fórmulas do BD formam o prefixo da sequência são rotuladas 1: $\varphi 1$, 2:$\varphi 2$, ..., k: φk;
2. Se a última fórmula da sequência é ρ.r: φi (onde ρ pode ser a sequência vazia), então a próxima fórmula rotulada será:
   2.1. $\rho.r.1$: φ j se φi é uma suposição;
   2.2. $\rho.r+1$: φ j se φ j foi obtida pela aplicação de regras do grupo I a fórmulas no escopo igual superior a σ, onde $\rho = \sigma.r$;
   2.3. $\sigma.s+1$: φ j se φ j foi obtida pela aplicação das regras do grupo II e $\rho = \sigma.s$;
   2.4. .n: φ é a última fórmula da sequência.

(ii) A fórmula φ é dita um **teorema do conjunto de fórmulas** BD, $BD \vdash \varphi$.

(iii) A fórmula φ é dita um teorema lógico se BD é vazio. OBS:Uma prova pode ser chamada algumas vezes de derivação.

Definição 2.2.4.2. :

1 Um conjunto de fórmulas $\Gamma = \{\alpha 1, \alpha 2, ..., \alpha n\}$ é **inconsistente** se somente se $\Gamma \vdash \beta \wedge \neg\beta$ para alguma fórmula β

2 Um conjunto de fórmulas de fórmulas $\Gamma = \{\alpha 1, \alpha 2, ..., \alpha n\}$ é **consistente** se somente se ele não é inconsistente.

Seguem agora abaixo exemplos de dedução natural.

Exemplo 2.2.4.0.1. $BD = \{A \wedge B, A \to C, C \to D\}$
Pergunta: D

1. $A \wedge B$ BD
2. $A \to C$ BD
3. $C \to D$ BD
4. A $\wedge E(1)$
5. C $\to E(2,4)$
6. D $\to E(3,5)$

Conforme pode-se observar pelo exemplo acima D é consequência lógica do BD.

Exemplo 2.2.4.0.2. $BD = \{A \to C, C \to D\}$
Pergunta: $A \to D$

1. $A \to C$ BD
2. $C \to D$ BD
3. $[A]^1$ Suposição
   - 3.1. C $\to E(3,1)$
   - 3.2. D $\to E(3.1,2)$
4. $A \to D$ $\to I(3.2,3)$

Exemplo 2.2.4.0.3. $BD = \{A \vee B, A \to C, B \to C\}$
Pergunta: C

1. $A \vee B$ BD
2. $A \to C$ BD
3. $B \to C$ BD
4. $[A]^1$ Suposição
   - 4.1. C $\to E(4,2)$
5. $[B]^2$ Suposição
   - 5.1. C $\to E(5,3)$
6. $C^{1,2}$ $\vee E(1,4.1,5.1)$

Exemplo 2.2.4.0.4. $BD = \{A \to C\}$
Pergunta: $\neg(A \wedge \neg C)$

1. $A \to C$ BD

2. $[\neg\neg(A \wedge \neg C)]^1$ Suposição
   - 2.1. $(A \wedge C)$ $\neg E(2)$
   - 2.2. A $\wedge E(2.1)$
   - 2.3. $\neg C$ $\wedge E(2.1)$
   - 2.4. C $\rightarrow E(2.2, 1)$
   - 2.5. $C \wedge \neg C$ $\wedge I(2.3, 2.4)$
   - 2.6. $\perp$ $ABS(2.5)$
3. $\neg(A \wedge \neg C)^1$ $RAAI(2, 2.6)$

2.2.5 Método do Tableaux

Nesta seção será mostrado o método do Tableaux apresentado por (SMULLYAN, 1968). Este método consiste em deduções feitas por refutação, i.e., caso se queira deduzir α a partir de um banco de fórmulas BD, BD $\vdash \alpha$, se parte da negação de α e através da aplicação sucessiva de regras, tenta-se chegar no absurdo. As deduções tem forma de árvore. Assim como na dedução natural 2.4.1 a principal diferença entre o Tableaux da Lógica Proposicional com o Tableaux da Lógica de Primeira Ordem será o repertório de regras disponíveis.

Nesta seção serão apresentadas as regras e exemplos do Tableaux da Lógica Proposicional.

R_1

$$\frac{\alpha \wedge \beta}{\begin{array}{c}\alpha \\ \beta\end{array}}$$

R_2

$$\frac{\alpha \vee \beta}{\alpha \quad \beta}$$

R_3

$$\frac{\alpha \rightarrow \beta}{\neg\alpha \quad \beta}$$

R_4

$$\frac{\neg\neg\alpha}{\alpha}$$

R_5

$$\frac{\neg(\alpha \wedge \beta)}{\neg\alpha \quad \neg\beta}$$

R_6

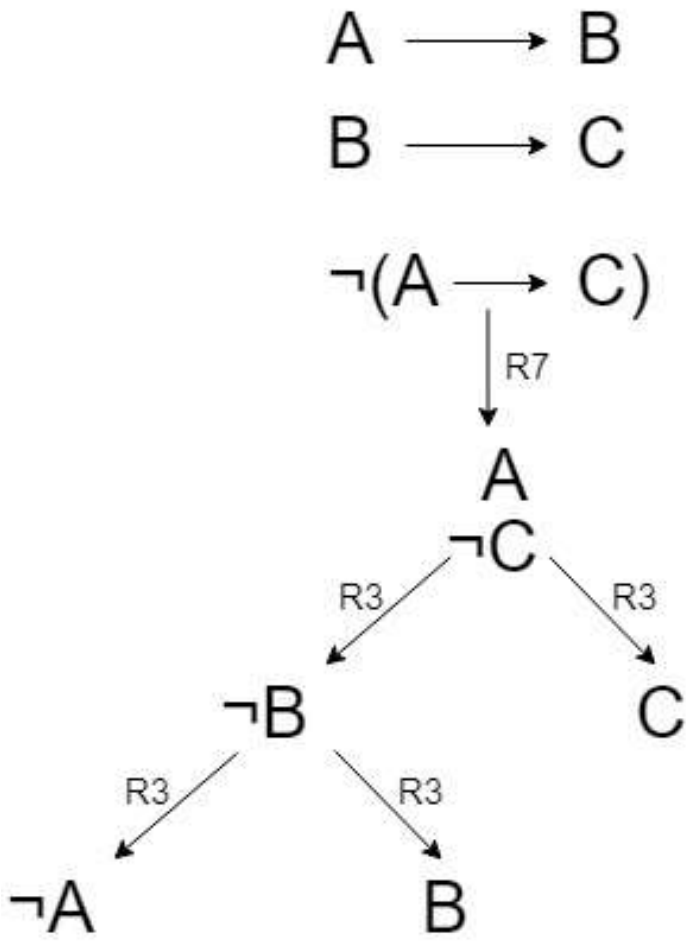
$$\frac{\neg(\alpha \vee \beta}{\begin{array}{l}\alpha\\ \beta\end{array}}$$

R_7

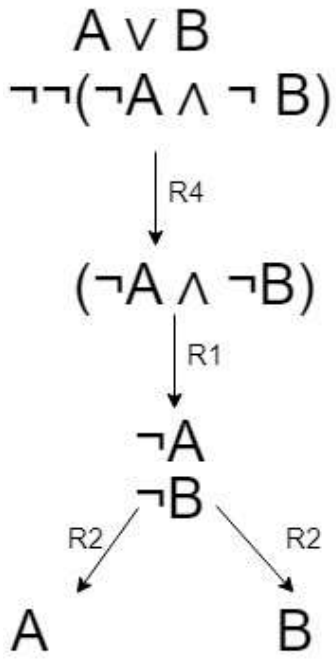
$$\frac{\neg(\alpha \rightarrow \beta)}{\begin{array}{l}\alpha\\ \neg\beta\end{array}}$$

Seguem abaixo alguns exemplos para mostrar a aplicação das regras:

Exemplo 2.2.5.0.1. $\{(A \to B), (B \to C)\} \vdash (A \to C)$



Exemplo 2.2.5.0.2. $\{(A \vee B)\} \vdash \neg(\neg A \wedge \neg C)$



2.2.6 Método de Resolução

O método de resolução consiste em um outro método para resolver o grande dilema do curso de lógica, saber se dado um BD, a pergunta é consequência lógica deste BD. O método será apresentado em forma de algoritmo com poucos passos e para se compreender este algoritmo é necessário antes apresentar alguns um conceitos.

Definição 2.2.6.1. Átomo e literal

- Uma fórmula atômica ou átomo é qualquer símbolo proposicional. Ex: A, B,...Z.
- Um literal é um átomo ou sua negação. Ex: $A, \neg A, B, \neg C$.

O próximo conceito a ser apresentado é denominado **Forma Normal Conjuntiva**. Uma fórmula α está na forma normal conjuntiva se e somente se α tem a seguinte forma:

$$\alpha = D_1 \wedge D_2 \wedge ... \wedge D_n$$

onde cada $D_i = (Q_1 \vee Q_2 \vee ... \vee Q_m)$, $1 \leq i \leq n$ e Q_i, $1 \leq j \leq m$, são literais, isto é, cada D_i é uma disjunção de literais. E α é uma conjunção de disjunções de literais.

Algoritmo 1 Algoritmo de conversão de fórmulas para a FNC

Entrada: Uma fórmula

Saída: Fórmula convertida em FNC

1. Se houver conectivo $\rightarrow$ então:
2. Se $\alpha \rightarrow \beta$ então $(\neg\alpha \vee \beta)$
3. Se $\neg(\alpha \rightarrow \beta)$ então $(\alpha \wedge \not\beta)$
4. Mover a negação para o interior da fórmula, usando as seguintes regras:
5. Se $\neg\neg\alpha$ então α
6. Se $\neg(\alpha \wedge \beta)$ então $(\neg\alpha \vee \neg\beta)$
7. Se $\neg(\alpha \vee \beta)$ então $(\neg\alpha \wedge \neg\beta)$
8. Mover as conjunções para o exterior da fórmula usando:
9. Se $\alpha \vee (\beta \wedge \gamma)$ então $(\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
10. Se $(\alpha \wedge \beta) \vee \gamma$ então $(\alpha \vee \gamma) \wedge (\beta \vee \gamma)$

Com o algoritmo para conversão para FNC apresentado, agora pode-se apresentar o algoritmo da resolução propriamente dito. Segue:

Algoritmo 2 Algoritmo de resolução

Entrada: Um banco de dados e uma pergunta

Saída: Contradição, indicando que a pergunta é consequência lógica do BD, ou apenas fórmulas simplificadas caso não seja possível responder a pergunta.

1. Passar o BD para FNC e quebrar as conjunções em Cláusulas;
2. Negar a pergunta, passá-la para FNC e quebrar as conjunções em Cláusulas
3. Juntas as cláusulas obtidas nos passos 1 e 2 e aplicar as regras até obter a cláusula vazia $\square$ (contradição).

Regras de Resolução

$$\frac{L_1, ..., L_i, \alpha, L_{i+1}, ..., L_n \quad M_1, ..., M_j, \not\alpha, M_{j+1}, ..., M_k}{L_1, ..., L_n, M_1, ..., M_k}$$

$$\frac{L_1, ..., L_i, \alpha, L_{i+1}, ..., L_j, \alpha, L_{j+1}, ..., L_n}{L_1, ..., L_i, \alpha, L_{i+1}, ..., L_j, L_{j+1}, ..., L_n}$$

Com os algoritmos devidamente apresentados, agora pode-se mostrar a aplicação da resolução na lógica proposicional. No entanto como toda a teoria se baseia-se somente na apresentação do algoritmo, nesta seção serão apresentados apenas exemplos. Seguem os exemplos abaixo:

Exemplo 2.2.6.0.1. $BD = \{A \vee B, A \rightarrow C, B \rightarrow C\}$ Pergunta: C

1. $A \vee B$ BD em FNC
2. $\not A \vee C$ BD em FNC
3. $\neg B \vee C$ BD em FNC
4. $\neg C$ Negação da pergunta
5. $\neg B$ (3,4)
6. $\neg A$ (2,4)
7. A (5,1)
8. $\square$ (6,7)

Exemplo 2.2.6.0.2. $BD = \{A \vee B, A \rightarrow B, B \rightarrow A\}$ Pergunta: $A \wedge B$

1. $A \vee B$ BD em FNC
2. $\neg A \vee B$ BD em FNC
3. $\neg B \vee A$ BD em FNC
4. $\neg A \vee \neg B$ Negação da pergunta
5. $\neg B \vee \neg B$ (3,4)
6. $\neg B$ (5)
7. A (6,1)
8. $\neg A$ (6,2)
9. $\square$ (7,8)

2.3 LINGUAGEM DA LÓGICA CLÁSSICA DE 1ª ORDEM

2.3.1 Linguagem

A linguagem de primeira ordem é uma linguagem formal que contem a linguagem proposicional, acrescida de outros elementos que garantem um grande poder de representação. Estes novos elementos são, **variáveis**, **constantes**, **funções** e **Tabelas(Predicados)**.

Assim como foi feito na linguagem proposicional, serão apresentados o alfabeto e a gramática da linguagem de primeira ordem. Na lógica de primeira ordem, o alfabeto é composto pelos seguintes elementos:

- Símbolos lógicos:
  1. **Conectivos lógicos**: $\neg$ (negação, lê-se: NÃO), $\wedge$ (conjunção, lê-se: E), $\vee$(disjunção, lê-se:OU), $\rightarrow$(implicação, lê-se: SE...ENTÃO...), $\forall$ (quantificador universal, lê-se: PARA TODO), $\exists$ (quantificador existencial, lê-se: EXISTE).
  2. **Símbolos de pontuação**:(,);
  3. **Conjunto enumerável e infinito de variáveis**: $V = \{v1, v2, ...\}$ ou conforme a convenção $V = \{x, y, z, ...\}$
  4. **Símbolos de verdade**: $verdadeiro, falso$
- Símbolos não-lógicos:
  1. **Conjunto enumerável de constantes**: $C = \{c1, c2, ...\}$ ou conforme a convenção $C = \{a, b, c, ...\}$
  2. **Conjunto enumerável de símbolos de função**: $F = \{f1, f2, ...\}$ ou conforme a convenção $F = \{F, G, H, ...\}$
     A cada símbolo funcional está associado um número inteiro $n > 0$, chamado de aridade.
  3. **Conjunto enumerável de símbolos predicativos (Predicados)**: $P = \{P1, P2, ...\}$ ou conforme a convenção $P = \{A, B, C, P, U, ...\}$
     A cada símbolo predicativo está associado um número $n > 0$, chamado aridade

Dada a introdução de vários novos termos, convém descrever o cada um significa.

- **Variáveis**: *representam elementos quaisquer do domínio.*
- **Constantes**: *dão nome a elementos particulares do domínio.*
- **Funções**: *representam operações sobre elementos do domínio.*
- **Predicados**: *representam propriedades ou relações entre elementos do domínio.*
- $\forall$ **Quantificador universal**: *"para todo elemento do domínio".*
- $\exists$ **Quantificador existencial**: *"existe ao menos um indivíduo no domínio".*

Para se descrever a gramática na linguagem de primeira ordem, diferente da linguagem proposicional onde só se trabalha sobre as fórmulas, haverá aqui o conceito de **termos**. Seguem abaixo as representações gramaticais da linguagem de primeira ordem:
Os **termos** da linguagem de 1ª ordem são definidos recursivamente como:

- *toda variável e constante é um termo;*
- *se t1, t2,...,tn são termos e f um símbolo funcional de aridade n, f(t1,t2,...,tn) é um termo;*
- *nada mais é termo.*

As **fórmulas** da lógica de 1ª ordem são definidas recursivamente como:

- *Se P é um predicado de aridade n e t1,t2,..,tn são termos, então P(t1,t2,...,tn) é uma fórmula chamada* ***fórmula atômica****;*
- *Se α e β são fórmulas, então $(\neg\alpha)$, $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, $(\alpha \rightarrow \beta)$ também são fórmulas*
- *Se α é uma fórmula e x é uma variável, então $\forall x\ \alpha$ e $\exists x\ \alpha$ também são fórmulas;*
- *Nada mais é fórmula*

Alternativamente podemos escrever definir gramaticalmente a linguagem de primeira ordem por meio da notação BNF.
Termos:

$$t ::= \ x \mid c \mid f(t_1, ..., t_n)$$

Fórmulas:

$$\alpha ::= P(t_1, ..., t_n) \mid (\alpha_1 \wedge \alpha_2) \mid (\alpha_1 \vee \alpha_2) \mid (\alpha_1 \rightarrow \alpha_2) \mid \neg\alpha \mid \forall x\alpha(x) \mid \exists x\alpha(x)$$

onde P é um símbolo predicativo n-ários e $t_1, ..., t_n$ são termos.

Definição 2.3.1.1. Dizemos que uma variável x ocorre livre em uma fórmula α se somente se:

i α é uma fórmula atômica e x ocorre em α;

ii α é uma fórmula da forma $\beta \wedge \gamma$, $\beta \vee \gamma$, $\beta \rightarrow \gamma$ e x ocorre livre em β ou γ;

iii α é uma fórmula da forma β e x ocorre livre em β;

iv α é uma fórmula da forma $\forall y\beta$ ou $\exists y\beta$ e x ocorre livre em β e $x \neq y$.

Definição 2.3.1.2. Uma fórmula α é uma sentença (ou uma fórmula fechada) se somente se α não tem nenhuma variável ocorrendo livre.

Definição 2.3.1.3. Seja α uma fórmula, x uma variável e t um termo. Pela substituição de x por t em $\alpha(\alpha(x/t))$ entendemos a expressão resultante da troca de todas as ocorrências livres de x por t.

Definição 2.3.1.4. Denotaremos por: $\alpha(x_1, x_2, ...x_n)$ $(x_1/t_1, x_2/t_2, ..., x_n/t_n)$ a substituição simultânea (paralelo) de todas as ocorrências livres de $x_1, ..., x_n$ por $t_1, ..., t_n$ respectivamente.

Definição 2.3.1.5. Uma variável x é substituível em uma fórmula α por um termo t se, para cada variável y, ocorrendo em t, não existe nenhuma subfórmula de α da forma $\forall y\beta$ ou $\exists y\beta$ onde x ocorre livre em β. O que se quer evitar com esta condição é que o quantificador $\forall$ y ou $\exists$ y capture alguma variável de t.

2.3.2 Semântica

A linguagem de primeira ordem possui um poder de representatividade muito grande, de modo que detalhar de maneira técnica sua semântica e suas propriedades exigem muitas definições, como foge do escopo deste trabalho explorar toda a complexidade da linguagem de primeira ordem, serão apresentadas as definições necessárias na durante a explicação de cada método em 2.3.3 com o propósito de mostrar seu funcionamento e posteriormente discutir suas implementações. Conforme (BEDREGAL; ACIóLY, 2002), na linguagem proposicional toda proposição era composta de proposições atômicas, usando os conectivos lógicos $\neg, \wedge, \vee, \rightarrow\leftrightarrow$. Para fornecer uma interpretação para uma proposição, tudo que precisávamos saber era se as proposições atômicas eram verdadeiras ou falsas. Podíamos listar todas as possíveis interpretações usando a tabela verdade. Agora a situação é mais complexa. Podemos expressar relações que são verdadeiras para certos indivíduos e não para outros, em vez de simplesmente estabelecer que é verdadeira ou falsa. Para determinar a veracidade ou falsidade de uma fórmula em uma interpretação

precisamos saber que valores as variáveis podem tomar, que elementos as constantes interpretam, e que funções e relações iriam representar os símbolos de função e de relação, respectivamente.

2.3.3 Consequências Lógicas de Primeira Ordem

2.3.3.1 Método de Decução natural

No contexto de lógica de primeira ordem, a dedução natural funciona essencialmente da mesma maneira que na lógica proposicional, acrescido de novas regras. Para apresentá-las convém se ter em mente a definição de variável substituível apresentada em **Definição 2.3.1.5**.

Regras do Quantificador Universal $\forall$

$\forall - \mathbf{I}$

$$\frac{\alpha}{\forall x \alpha(a/x)}$$

$\forall - \mathbf{E}$

$$\frac{\forall x \alpha}{\alpha(x/t)}$$

Regras para o Quantificador Existencial $\exists$
$\exists - \mathbf{I}$

$$\frac{\alpha(a)}{\exists x \alpha(a/x)}$$

$\exists - \mathbf{E}$

$$\frac{\exists x \alpha(x) \qquad \begin{array}{c}[\alpha(a)]^i \\ \vdots \\ \theta\end{array}}{\theta^i}$$

As regras da dedução natural da lógica de primeira ordem possuem algumas condições para poderem serem aplicadas, seguem elas:

- $\forall$ - I, para a inclusão do $\forall$ a não pode ocorrer em nenhuma fórmula do BD e nem em nenhuma suposição em aberto.
- $\forall$ - E, para a exclusão do para todo, x deve ser substituível em α por $\mathbf{t}$.
- $\exists$ - I, para a inclusão do $\exists$ não existem restrições.
- $\exists$ - E, para a exclusão do $\exists$ a não ocorre nem em θ, nem no BD e nem em qualquer suposição na qual θ depende a não ser $\alpha(x/a)$.

Seguem abaixo alguns exemplos que irão ajudar na discussão dos resultados.

Exemplo 2.3.3.1.1. $BD = Vazio$
Pergunta: $\vdash (\forall x \forall y P(x,y) \rightarrow \forall y \forall x P(x,y))$

1. $[\forall x \forall y P(x,y)]^1$ Suposição
   - 1.1. $\forall y P(a,y)$ $\forall E - (1)$
   - 1.2. $P(a,b)$ $\forall E - (1.1)$
   - 1.3. $\forall x P(x,b)$ $\forall I - (1.2)$
   - 1.4. $\forall y \forall x P(x,y)$ $\forall I - (1.3)$
2. $(\forall x \forall y P(x,y) \rightarrow \forall y \forall x P - (x,y))$ $\rightarrow I - (1, 1.4)$

Exemplo 2.3.3.1.2. $BD = Vazio$
Pergunta: $\vdash ((\forall x(Q(y) \rightarrow P(X))) \rightarrow (Q(y) \rightarrow \forall x P(x)))$

1. $[\forall x(Q(y) \rightarrow P(x))]^1$ Suposição

1.1. $(Q(y) \to P(a)) \qquad \forall E-(1)$

1.2. $[Q(y)]^2 \qquad$ Suposição

1.2.1. $P(a) \qquad \to -E(1.1, 1.2)$

1.2.2. $\forall x P(x) \qquad \forall - I(1.2.1)$

1.3. $(Q(y) \to \forall P(x))^2 \qquad \to I-(1.2, 1.2.2)$

2. $((\forall x(Q(y) \to P(X))) \to (Q(y) \to \forall x P(x))) \qquad \to I(1, 1.3)$

Exemplo 2.3.3.1.3. $BD = \{(\forall x \forall y P(x, y)), (\forall x \forall y(P(x, y) \to Q(x) \wedge R(y))), ((\exists x R(x) \wedge \forall x Q(x)) \to (\exists x(S(x) \wedge T(x))))\}$
Pergunta: $\vdash (\exists x T(x))$

1. $(\forall x \forall y P(x,y))$
2. $(\forall x \forall y (P(x,y) \rightarrow Q(x) \wedge R(y)))$
3. $((\exists x R(x) \wedge \forall x Q(x)) \rightarrow (\exists x (S(x) \wedge T(x))))$
4. $\forall y P(a,y) \qquad \forall - E(1)(x/a)$
5. $yP(a,b) \qquad \forall - E(4)(y/b)$
6. $(\forall y (P(a,y) \rightarrow Q(a) \wedge R(y))) \qquad \forall - E(2)(x/a)$
7. $(y(P(a,b) \rightarrow Q(a) \wedge R(b))) \qquad \forall - E(2)(y/b)$
8. $Q(a) \wedge R(b) \qquad \rightarrow -E(5,7)$
9. $Q(a) \qquad \wedge - E(8)$
10. $\forall x Q(x) \qquad \forall - I(9)$
11. $R(b) \qquad \wedge - E(8)$
12. $\exists R(x) \qquad \exists - I(11)$
13. $\exists R(x) \wedge \forall x Q(x) \qquad \wedge - I(10,12)$
14. $\exists x (S(x) \wedge T(x)) \qquad \rightarrow -E(13,3)$
15. $[(S(a) \wedge T(a))]^1 \qquad$ Suposição

    15.1. $T(a) \qquad \wedge - E(15)$

    15.2. $\exists x T(x) \qquad \exists - I(15.1)$
16. $\exists x T(x)^1 \qquad \exists E(14,15,15.1)$

2.3.3.2 Método do Tableaux

Para o tableaux de lógica de primeira ordem, têm-se a mesma estrutura do tableaux da lógica proposicional, acrescido de novas regras com os quantificadores. Seguem as regras:
R_8

$$\frac{\forall x \alpha}{\alpha(x/t)}$$

R_9

$$\frac{\neg \forall x \alpha}{\neg \alpha(x/t)}$$

R_{10}

$$\frac{\exists x \alpha}{\alpha(x/t)}$$

R_{11}

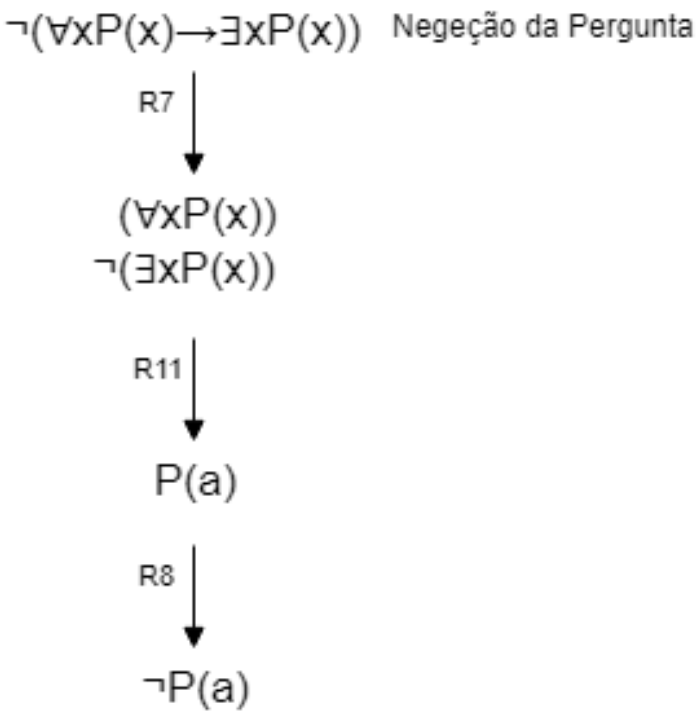
$$\frac{\neg \exists x \alpha}{\neg \alpha(x/t)}$$

No entanto existem algumas condições para a aplicação destas regras, seguem abaixo:

- **Condição para R_8:** t é um termo qualquer;
- **Condição para R_9:** t é um termo novo no ramo em que ele aparece Tableaux;
- **Condição para R_{10}:** t é um termo novo no ramo em que ele aparece Tableaux;
- **Condição para R_{11}:** t é um termo qualquer;
- **Condição:** As regras R_8 e R_{11} podem ser usadas mais de uma vez nos mesmos ramos do Tableaux.

Seguem abaixo alguns exemplos para mostrar a aplicação das regras:

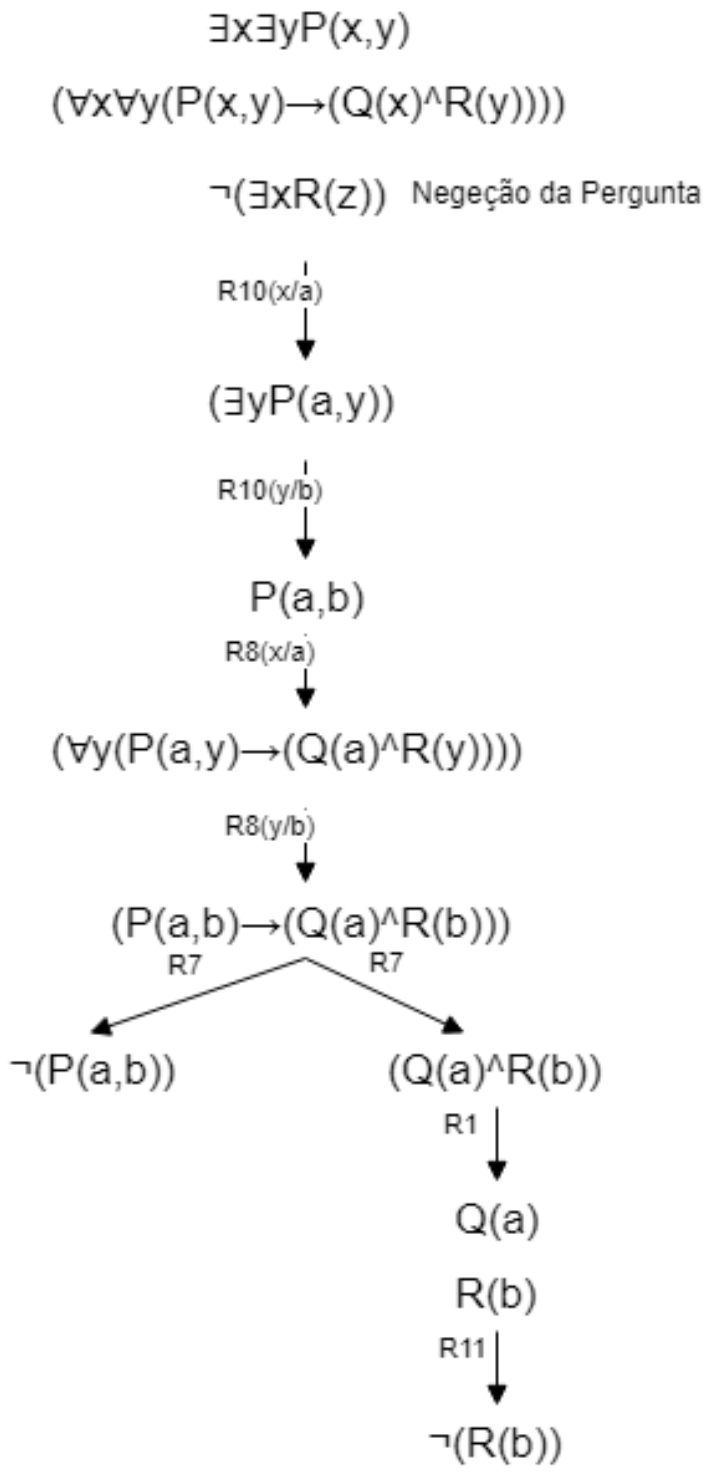
Exemplo 2.3.3.2.1. $\vdash (\forall x P(X) \rightarrow \exists x P(x))$



Exemplo 2.3.3.2.2. $\{(\exists x \exists y P(x,y)), (\forall x \forall y (P(x,y) \rightarrow (Q(x) \wedge R(y))\} \vdash (\exists z R(z))$

2.3.3.3 Método da Resolução

No curso de lógica não é apresentado o método da resolução para lógica de primeira ordem.



2.3.4 Semântica

Um alfabeto de primeira ordem introduz símbolos de função e de predicado, isto é, nomes de funções e de predicados. Esses símbolos ou nomes podem vir a ser interpretados de muitas formas diferentes, no sentido em que a cada um desses nomes se pode fazer corresponder uma certa função (total) ou predicado. Quando se faz corresponder a cada símbolo de função uma certa função e a cada símbolo de predicado um certo predicado

(propriedade) está a definir-se uma estrutura de interpretação. As funções e predicados são definidos à custa de conjuntos, pelo que as estruturas de interpretação incluem também conjuntos. Na lógica de primeira ordem que é objeto de estudo nesta seção, os símbolos de função e de predicado são interpretados como funções e predicados envolvendo apenas um conjunto. Assim, cada estrutura de interpretação inclui apenas um conjunto, designado domínio o ou universo de interpretação. A interpretação de um termo numa estrutura de interpretação é um elemento desse domínio e a interpretação de uma fórmula é uma afirmação sobre elementos desse domínio.
No entanto, se se considerar apenas o fragmento proposicional do alfabeto, não existem termos e só se podem construir fórmulas proposicionais. Havendo só fórmulas proposicionais não é necessário considerar um domínio, basta atribuir a cada símbolo proposicional um valor booleano. Assim, a semântica do fragmento proposicional é mais simples que no caso geral.
Em geral, para sabermos se uma fórmula é verdadeira ou falsa, nós precisamos saber o universo e interpretar cada símbolo não-lógico neste universo e dar valor as variáveis livres.

1. Interpretar variáveis livres e constantes em elementos do domínio.
2. Interpretar predicados em relações entre elementos do domínio.
3. Interpretar funções em funções sobre o domínio.

Definição 2.3.4.0.1. Definimos uma **interpretação** como sendo um par ordenado $L =< D, I >$ onde D é um conjunto não-vazio de indivíduos chamado **domínio**. E I é uma função chamada de **função de interpretação**, definida como:

1. I associa a cada variável livre x um elemento do domínio $d^I \in D$. $I(x) = d^I$
2. I associa a cada constante c, um elemento do domínio $c^I \in D$. $I(c) = c^i$
3. I associa a cada símbolo funcional n-ário f uma função n-ária $f^I : D^n \to D$ tal que $I(f(t_1, ..., t_n)) = f^I(I(t_1), ..., I(t_n))$, onde $t_1, ..., t_n$ são termos.
4. I associa a cada símbolo predicativo n-ário P uma relação n-ária sobre D. $I(P) = P^I$, $P^I \subseteq D^n$, ie, $P^I \subseteq D \times D \times, ..., \times D$, n vezes

Definição 2.3.4.0.2. Seja L uma linguagem de primeira ordem e α e β, fórmulas de L,$t_1, ..., t_n$ termos, P um símbolo predicativo n-ário e $< D, I >$ uma interpretação. Definimos a função de avaliação de fórmulas de L como:
$V_L : W \to \{V, F\}$, onde W é o conjunto de fórmulas, tal que:

1. $V_L(P(t_1, ..., t_n)) = V$ se somente se $< I(t_1), ..., I(t_n) > \in P^I$. F caso contrário.
2. $V_L(\neg\alpha) = V$ se $V_L(\alpha) = F$. F caso contrário.

3. $V_L(\alpha \wedge \beta) = V$ se $V_L(\alpha) = V$ e $V_L(\beta) = V$. F caso contrário.

4. $V_L(\alpha \vee \beta) = V$ se $V_L(\alpha) = F$ e $V_L(\beta) = F$. V caso contrário.

5. $V_L(\alpha \rightarrow \beta) = F$ se $V_L(\alpha) = V$ e $V_L(\beta) = F$. V caso contrário.

6. $V_L(\forall x\alpha) = V$ se somente se para todo $d \in D$, se $I(x) = d$ então $V_L(\alpha) = V$. F caso contrário.

7. $V_L(\exists x\alpha) = V$ se para algum $d \in D$ e $V_L(\alpha) = V$. F caso contrário.

Definição 2.3.4.0.3. Seja L uma linguagem de 1ª ordem. I uma interpretação para L,Γ um conjunto de fórmulas de L e α uma fórmula.

1. I **satisfaz** $\alpha(\models_I \alpha)$ se somente se $V_I(\alpha) = V$;

2. **satisfaz** Γ se somente se satisfaz cada membro de Γ;

3. Γ é **satisfatível** se somente se existe uma interpretação I que satisfaça Γ;

4. α é **válida** $(\models \alpha)$ se somente se para toda interpretação I, $\models_I \alpha$, i.e., $V_I(\alpha) = V$ para todo I;

5. Γ **implica logicamente** em $\alpha(\Gamma \models \alpha)$ se somente se para toda interpretação I, se I satisfaz Γ, então I satisfaz α;

6. Γ é **insatisfatível** se somente se Γ não é satisfatível, i.e., não existe uma interpretação I que satisfaz Γ;

7. Uma interpretação I que satisfaz Γ é dita **modelo** para Γ.

Exemplo 2.3.4.0.1. $\forall x(P(x) \rightarrow E(x))$
Usaremos como domínio $D = \{0, 1\}$, que possui uma quantidade de valores suficientes para garantir que o teste seja completo, pode-se usar mais valores, porém apenas iriam aumentar o trabalho de análise da prova para produzir um resultado o qual este domínio já permite produzir.
Interpretação que não satisfaz:
Para está fórmula ser falsa, a única maneira é se ter $V \rightarrow F$ em pelo menos um conjunto de relações e assim por consequência o $\forall$ resultará F já que nem todas as possibilidades resultarão em V, assim podemos usar as seguintes relações: $P^I = \{< 0 >\}$ e $E^I = \{< 1 >\}$. Com isto, para os valores $P(0) \rightarrow E(0)$, teremos $V \rightarrow F$ e por consequência, uma das possibilidades do $\forall$ resultará em F, não satisfazendo portanto a fórmula.
Interpretação que satisfaz:
Para satisfazer pode-se usar um caminho fácil. As duas relações como vazias, assim segue que:$P^I = \{<>\}$ e $E^I = \{<>\}$. Que por consequência nos dará como possibilidades apenas $F \rightarrow F$ que sempre será verdadeiro, satisfazendo portanto a fórmula.

Exemplo 2.3.4.0.2. $\forall(P(x) \wedge \exists y Q(x,y))$

Não Satisfaz	Satisfaz
$D = \{0,1\}$	$D = \{0,1\}$
$P^I = \{< 0 >\}$	$P^I = \{< 0 >, < 1 >\}$
$Q^I = \{< 0,1 >\}$	$Q^I = \{< 0,1 >, < 1,0 >\}$

Exemplo 2.3.4.0.3. $\forall x(P(x,y) \rightarrow (Q(x) \vee R(c))$

Não Satisfaz	Satisfaz
$D = \{0,1\}$	$D = \{0,1\}$
$y^I = 0$	$y^I = 0\}$
$c^I = 0$	$c^I = 0\}$
$P^I = \{< 1,1 >\}$	$P^I = \{< 0,0 >, < 1,0 >\}$
$Q^I = \{< 1 >\}$	$Q^I = \{<>\}$
$R^I = \{< 0 >\}$	$Q^I = \{< 1 >\}$

3 CONCEITOS E TECNOLOGIAS EMPREGADAS

3.1 WEB SERVICE E APLICAÇÃO

Conforme mencionado no capítulo 1, a proposta do projeto é o desenvolvimento da LogicAPI: um *Web Service* que seja capaz de verificar consequências lógicas através de diferentes métodos, descritos em 2, além de uma aplicação de usuário que consuma os dados recebidos. Portanto, entender o que é um *Web Service* torna-se fundamental para o entendimento do projeto proposto.

O uso da tecnologia tem revolucionado a forma como o ser humano consome produtos, serviços e informações, e devido à grande popularização da internet é cada vez mais necessário uma forma de fazer com que aplicações das mais variadas arquiteturas e em diferentes localizações conversem entre si, possibilitando a transposição de barreiras geográficas e tecnológicas (RIBEIRO; FRANCISCO, 2016).

Pensando nesse desafio apresentado, podemos definir *Web Services* como componentes de software que, por meio de padrões previamente estabelecidos, fornecem serviços específicos e promovem trocas de informações entre sistemas, independentemente das arquiteturas, tecnologias ou linguagens de programação utilizadas na construção das aplicações envolvidas (PAUL; DEITEL, 2010).

Outra definição para *Web Service* o coloca como um componente de negócio que fornece uma funcionalidade reutilizável para clientes, ou consumidores, e pode ser pensado como um componente com acessibilidade verdadeiramente global - se houver direitos de acesso apropriados (SHARP, 2011).

Com esses conceitos em mãos, é possível avaliar a importância dos *Web Services* para o contexto em que vivemos, percebendo que a ferramenta é poderosa para solucionar os desafios de heterogeneidade e interoperabilidade (RIBEIRO; FRANCISCO, 2016).

É importante, ainda, atentar para o seu funcionamento. Conforme descrito em (OPENSOFT, 2016), uma aplicação solicita uma das operações disponíveis no *Web Service*, que então efetua o processamento e envia os dados para a aplicação que requereu a operação. A aplicação então recebe esses dados e faz a sua interpretação, convertendo-os para a sua linguagem própria.

Para se comunicarem, é necessário a utilização de uma linguagem intermediária, que seja entendida tanto pelo *Web Service* quanto pela aplicação requerente. Duas arquiteturas então surgem como principais (RIBEIRO; FRANCISCO, 2016) para permitir tal comunicação: *SOAP* (*Simple Object Access Protocol*) e *REST* (*Representational State Transfer*).

O protocolo *SOAP* utiliza-se do protocolo *HTTP* para transportar mensagens que possuem um documento *XML* em seu corpo, e estão associados a um documento *WSDL* (*Web*

Service Definition Language), que descreve as operações disponíveis no *Web Service* e a localização para acesso dessas operações (RIBEIRO; FRANCISCO, 2016) (OPENSOFT, 2016).

Já o protocolo *REST* é mais flexível, baseando-se e utilizando os verbetes *HTTP* (*GET, PUT, POST* e *DELETE*) e permitindo a utilização de vários formatos para a representação de dados, como *JSON*, *XML*, *RSS* e outros (OPENSOFT, 2016). Com isso, os *Web Services REST* são mais utilizados quando a performance é importante, já que tem a capacidade de transferir as informações diretamente pelo protocolo HTTP (OPENSOFT, 2016).

O *Web Service* implementado neste trabalho utiliza o protocolo *REST*, enviando, recebendo e processando arquivos JSON através dos verbetes *GET* e *POST*. Conforme mencionado anteriormente, uma aplicação foi construída para permitir interações dos usuários. A cada comando dado pelo usuário nesta aplicação, uma requisição com os dados necessários em formato *JSON* é enviada para o devido endereço do serviço implementado pelo *Web Service*, que por sua vez irá traduzir o arquivo *JSON*, fazer o processamento necessário e retornar uma resposta também em formato *JSON*. Então, a aplicação irá traduzir o arquivo *JSON* recebido e exibir a resposta para o usuário da forma mais apropriada.

O *Web Service* deste trabalho foi implementado na linguagem de programação *PHP* utilizando o *framework Laravel* e banco de dados *MySQL* para persistência de algumas informações importantes, como listas de exercícios, enquanto a aplicação de usuário foi desenvolvida utilizando tecnologias como *HTML, CSS* com *Bootstrap* e *Javascript*.

É importante destacar, entretanto, que o processamento de todos os métodos de lógica ocorre no *Web Service*, sendo a aplicação de usuário responsável apenas por permitir que o usuário se comunique com os serviços implementados e veja as respostas de uma forma amigável (*user-friendly*). Isso significa que futuras aplicações em quaisquer linguagem e arquitetura poderão ser criadas utilizando os métodos implementados neste trabalho para fazerem o processamento de expressões de lógica, desde que possuam suporte ao protocolo *HTTP*.

A seguir veremos uma breve descrição das tecnologias utilizadas para implementar o *Web Service* e a aplicação, bem como a motivação para a escolha de cada uma delas.

3.2 PHP

Segundo a documentação oficial, o *PHP* (um acrônimo recursivo para *PHP: Hypertext Preprocessor*) é uma linguagem de *script open source* de uso geral, muito utilizada, e especialmente adequada para o desenvolvimento *web* e que pode ser embutida dentro do *HTML* (PHP, 2018).

O que distingue o *PHP* de algo como o *JavaScript* interpretado no lado do cliente é que o código é executado no servidor, gerando o *HTML* que é então enviado para o

navegador. O navegador recebe os resultados da execução desse *script*, mas não sabe qual era o código fonte original (PHP, 2018).

Inicialmente, a ideia do projeto era utilizar ferramentas de *Java* para *Web* (*JPA, JSF* e *Hibernate*) para a produção do *Web Service*. Entretanto, devido as dificuldades de configuração e posteriormente de *deploy* do sistema para algum *host*,foi feita a escolha por *PHP* para ser a linguagem utilizada para a construção do *Web Service*.

Essa escolha é justificada pelo fato de que, ao mesmo tempo que o *PHP* é simples para um iniciante, oferece recursos avançados para um programador profissional (PHP, 2018).

Além disso, *PHP* é uma das linguagens mais utilizadas do mundo de acordo com diversos *rankings*. De acordo com (MISIRLAKIS, 2018) e conforme pode ser visto na figura 2, o *PHP* é uma das sete linguagens de programação com maior demanda de profissionais no mercado, sendo uma das duas que, entre as sete linguagens, obteve crescimento de demanda, com quase 8% de crescimento em relação ao ano anterior e quase 40% em relação à 2016 (PATEL, 2017) (BOUWKAMP, 2016).

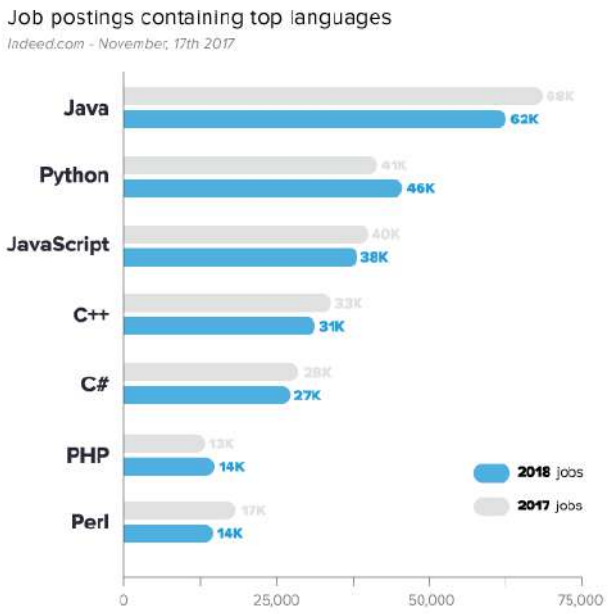


Figura 2 – Comparativo de demandas de cargos de trabalho entre 2017 e 2018

O *PHP* não desponta apenas como linguagem de programação com alta demanda de empregos no mercado de trabalho, mas também como uma das mais populares (PUTANO, 2018). Uma empresa de qualidade de *software* chamada TIOBE realiza anualmente uma pesquisa de popularidade de linguagens de programação, chamado de *TIOBE Index* (figura 3), onde este aponta o *PHP* como a sétima linguagem de programação mais popular, com um ligeiro crescimento de 0,63% em relação ao ano anterior (TIOBE, 2018).

Podemos observar a popularidade da linguagem também quando olhamos para grandes plataformas como o *GitHub* e o *StackOverflow*. Essas duas plataformas costumam

Aug 2018	Aug 2017	Change	Programming Language	Ratings	Change
1	1		Java	16.881%	+3.92%
2	2		C	14.966%	+6.49%
3	3		C++	7.471%	+1.92%
4	5	^	Python	6.992%	+3.30%
5	6	^	Visual Basic .NET	4.762%	+2.19%
6	4	v	C#	3.541%	-0.65%
7	7		PHP	2.925%	+0.63%
8	8		JavaScript	2.411%	+0.31%
9	-	^^	SQL	2.316%	+2.32%
10	14	^^	Assembly language	1.409%	-0.40%
11	11		Swift	1.384%	-0.44%
12	12		Delphi/Object Pascal	1.372%	-0.45%
13	17	^^	MATLAB	1.386%	-0.25%
14	18	^^	Objective-C	1.356%	-0.15%
15	10	vv	Ruby	1.182%	-0.78%
16	9	vv	Perl	1.175%	-0.82%
17	16	v	Go	0.996%	-0.65%
18	15	v	R	0.965%	-0.80%
19	13	vv	Visual Basic	0.922%	-0.89%
20	21	^	PL/SQL	0.702%	-0.51%

Figura 3 – TIOBE Index de Agosto de 2018

realizar mensurações e pesquisas com seus usuários para determinar, entre outros, quais são as linguagens de programação mais populares em seus meios. O *GitHub* dispõe de uma ferramenta de análise de tudo o que aconteceu na plataforma, chamado de *Octoverse*. Ela indica que em 2018 o *PHP* foi a quinta linguagem que mais teve requisições de *pull* para alteração de código, ficando a frente de linguagens como *C#* e *C++* (OCTOVERSE, 2018), o que pode ser visto na figura 4. Já o *StackOverflow* fez um *survey* em sua base de usuários e entre as 100.000 respostas, o *PHP* figura como a nona linguagem de programação mais popular, sendo utilizada por mais de 30% da comunidade (STACKOVERFLOW, 2018). O resultado do *survey* pode ser visto na figura 5.

Juntando todos esses fatores, podemos perceber que a escolha pela linguagem *PHP* para trabalhar com o *backend* da aplicação é certamente uma escolha responsável para o projeto. Só que mesmo escolhendo o *PHP*, outras decisões importantes precisaram ser tomadas, como por exemplo trabalhar puramente com *PHP* ou utilizar algum de seus muitos *frameworks*.

3.2.1 Laravel

Após decidir utilizar o *PHP* como linguagem de programação para o *Web Service*, a próxima decisão era escolher trabalhar ou não com um *framework* e, dependendo da escolha, qual *framework* utilizar.

Um *framework*, no contexto de desenvolvimento de *software*, é uma plataforma que auxilia na construção de aplicações, oferecendo uma fundação para o desenvolvimento de sistemas em determinada linguagem. A diferença entre usar ou não um *framework* está no fato de que o *framework* normalmente oferece um conjunto de classes, métodos e funções predefinidas para gerenciar e processar *inputs*, dispositivos de *hardware*, entre outras

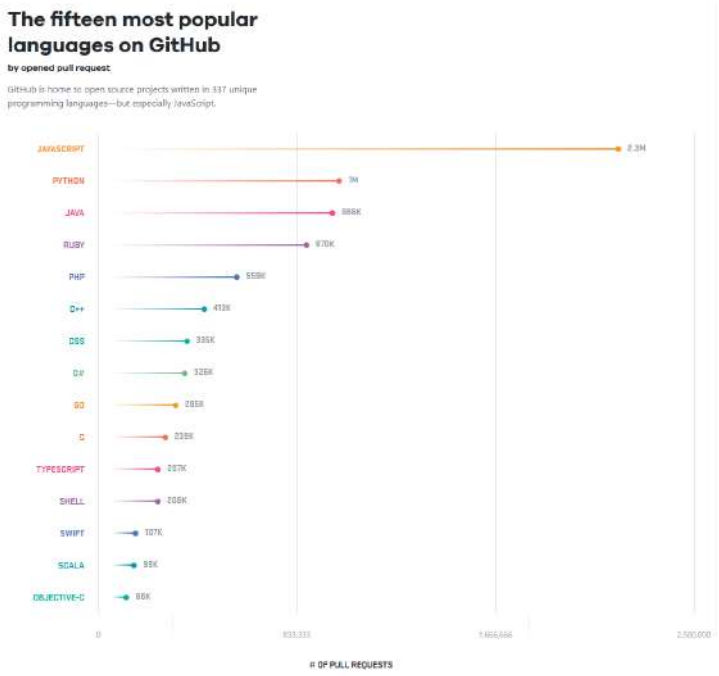


Figura 4 – Ranking das linguagens mais populares do GitHub em 2018

interações, proporcionando um conjunto de ferramentas úteis para o desenvolvimento de *software*(TECHTERMS, 2013).

Portanto, fica claro que utilizar-se de um *framework* para a construção da aplicação torna o desenvolvimento mais ágil, tendo em vista que diversos serviços que necessitariam de uma implementação já estão disponíveis nesse arcabouço proporcionado pelo *framework*. Para a construção do *Web Service* para solução de expressões lógicas, foi utilizado o *framework* para *PHP* chamado *Laravel*.

O *Laravel* é um *framework open source* de desenvolvimento para *PHP*, cujo principal objetivo é permitir que se trabalhe de forma estruturada e rápida (ROBERTO, 2013). Os próprios desenvolvedores do *Laravel* o definem como um *framework* de aplicações *web* com uma sintaxe expressiva e elegante, com o desafio de tornar o processo de desenvolvimento gratificante para o desenvolvedor, facilitando tarefas comuns na maioria dos projetos *web*, como autenticações, rotas, *sessions* e *caching* sem sacrificar a funcionalidade da aplicação (LARAVEL, 2018).

A escolha do *Laravel* se deu principalmente por duas razões, sendo a primeira delas a facilidade na configuração e utilização. Um *framework* como o *Laravel* é composto por uma série de bibliotecas de códigos, chamadas *libs*, com ferramentas e camadas de abstrações no qual pode-se utilizar-se para aumentar a segurança, facilidade de manutenção, velocidade de desenvolvimento e homogeneização do aplicativo (KAWAKAMI, 2013). Tais *libs* acabam por tornar o *framework* mais pesado e necessitando de atualizações constantes. Pensando nisso, o Laravel se utiliza de um gerenciador de dependências chamado *Composer*, para instalar e atualizar essas *libs*. Portanto, para começar a utilizar o *Lara-*

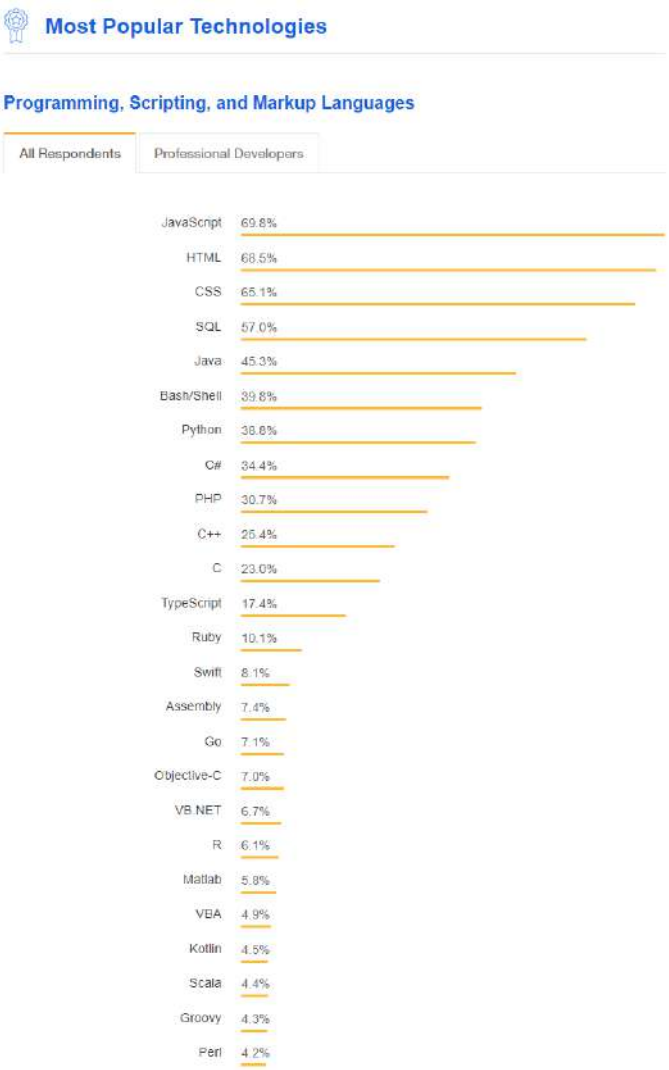


Figura 5 – Linguagens mais populares dos usuários do StackOverflow segundo survey

vel, o *Composer* faz o *download*, instalação e configuração das *libs* necessárias, deixando o *framework* pronto para utilização (KAWAKAMI, 2013).

A segunda razão pela escolha do *Laravel* é a sua popularidade. Mesmo sendo um *framework* relativamente novo comparado aos seus concorrentes, sendo lançado apenas em 2011, o *Laravel* se tornou o *framework PHP* mais utilizado, abrindo cada vez mais vantagem na liderança de popularidade. Isso fica claro quando analisamos a figura 6 que mostra o interesse em diferentes *frameworks* através das buscas no *Google* (CODERSEYE, 2018). Além disso, (MEDIUM, 2017) realizou um estudo em 2017 apontando o *Laravel* como o *framework PHP* mais utilizado do mercado, confome pode ser visto na figura 7.

Com isso, todas as tecnologias utilizadas para a construção do *Web Service* proposto foram cobertas, sendo este desenvolvido utilizando-se a linguagem de programação *PHP* e o *framework Laravel*. Essa é a parte que pode ser chamado de *backend* do sistema para ensino de lógica, ou seja, é no *Web Service* que estão implementadas as regras de negócio da

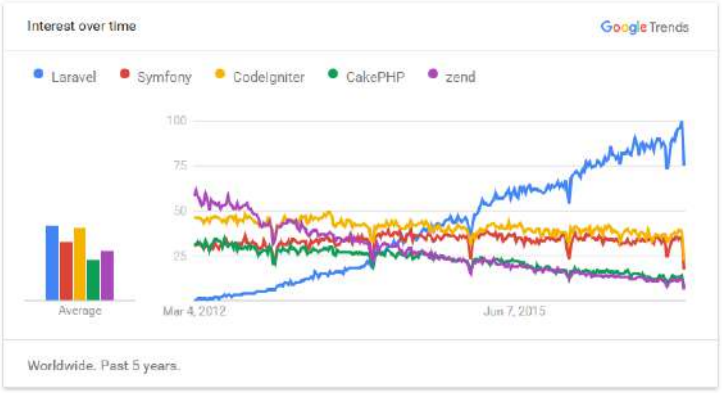


Figura 6 – Interesse nos principais frameworks PHP ao longo do tempo

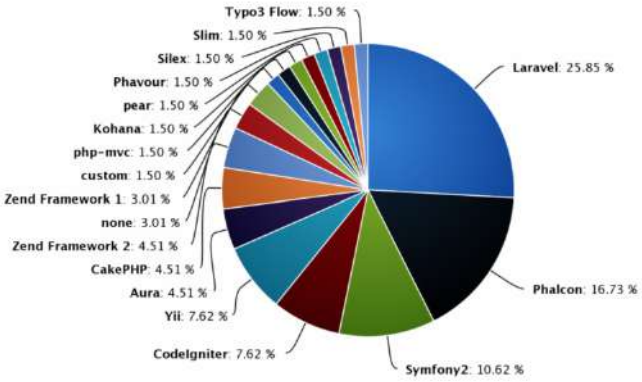


Figura 7 – Frameworks PHP mais utilizados em 2017

aplicação (TREINAWEB, 2017). A partir deste momento serão abordadas as tecnologias utilizadas para o desenvolvimento da aplicação do usuário, que pode ser classificada como o *frontend* da aplicação, ou seja, a interface da aplicação que interage diretamente com o usuário (TREINAWEB, 2017). Por fim, serão apresentadas as tecnologias utilizadas para persistência de alguns dados trocados entre *frontend* e *backend*.

3.3 HTML

Como uma aplicação *Web*, o *frontend* do sistema proposto se utiliza da linguagem de marcação *HTML* (*HyperText Markup Language*), que é a linguagem de marcação padrão para criação de páginas da *Web* (W3SCHOOLS, 2018g). O *HTML* estrutura as páginas *Web* usando marcações, que são as ações feitas ao texto dentro de uma *tag*, como por exemplo a tag $<i>$, que deixa todo o conteúdo dentro dela em itálico (SHANNON, 2018). Embora não seja obrigatório, uma boa prática consiste em fechar toda *tag HTML* aberta, adicionando uma barra logo após o colchete. Para exemplificar, a *tag* apresentada anteriormente que estrutura textos em itálico é fechada com: $</i>$.

O propósito de um navegador, como o *Chrome, Internet Explorer, Edge, Firefox, Safari* e *Opera*, é ler documentos *HTML*. Isto significa que os navegadores não exibem as *tags HTML*, mas as utilizam para renderizar o conteúdo na página (W3SCHOOLS, 2018g), conforme as estruturas que estão marcadas. Por essa razão o *HTML* é utilizado na interface do sistema, dado que a aplicação proposta é uma aplicação *Web*.

3.4 CSS

Apesar de representar toda a estrutura de um site, inclusive com algumas opções básicas de estilização, o *HTML* não tem o foco em apresentar para o usuário uma interface agradável, deixando esse papel para o *CSS* (*Cascading Style Sheets*), que é responsável por descrever o estilo de um documento *HTML* (W3SCHOOLS, 2018c). O *CSS* é responsável por descrever como elementos *HTML* são exibidos na tela, papel ou outra mídia, sendo capaz de controlar o layout de múltiplas páginas *Web* de uma vez só, economizando bastante trabalho (W3SCHOOLS, 2018c). Com isso, é possível escrever uma folha de estilos, armazenar em um arquivo separado com extensão *.css* e utilizar esta mesma folha para descrever estruturas comuns de diversas páginas da aplicação, padronizando o *layout*, economizando trabalho, minimizando erros e tornando a manutenção da estilização bem mais prática, de modo que a alteração no arquivo de estilos altera de uma só vez todas as páginas que o utilizam.

A aplicação de usuário proposta neste trabalho se utiliza de folhas de estilização para apresentar uma interface amigável ao usuário, ou seja, que a experiência do usuário ao utilizar o sistema seja prazerosa e de fácil manuseio e aprendizado (BISCHOF, 2012).

3.4.1 Bootstrap

Para ajudar no processo de tornar a interface amigável, foi utilizado o *framework* mais popular do mundo para criação de sites responsivos e mobile: o *Bootstrap* (BOOTSTRAP, 2018b). O *Bootstrap* é um pacote de ferramentas *open source* para o desenvolvimento com *HTML, CSS* e *Javascript* (BOOTSTRAP, 2018a), de forma a facilitar o desenvolvimento do *frontend* de aplicações *Web* (W3SCHOOLS, 2018b). Esse objetivo é atingido com a inclusão de *templates* baseados em *HTML* e *CSS* para tipografia, formulários, botões, tabelas, navegação, modais, entre outros elementos (W3SCHOOLS, 2018b), dando também a habilidade de criar *layouts* responsivos, isto é, *layouts* que se ajustam automaticamente para serem bem apresentados em todos os dispositivos, desde pequenos telefones até a grandes computadores (W3SCHOOLS, 2018b).

O *Bootstrap* oferece ferramentas também para *JavaScript*, porém neste trabalho foi utilizado apenas para estilização das página através de sua biblioteca *CSS*. Para o *JavaScript* foram utilizadas outras bibliotecas e *frameworks* que serão descritos na seção seguinte.

3.5 JAVASCRIPT

Segundo (W3SCHOOLS, 2018d), *JavaScript* é a linguagem de programação do *HTML* e da *Web*. *JavaScript* é a linguagem de programação usada para tornar páginas *Web* interativas (CHAPMAN, 2018), onde a diferença para o *HMTL* é que o *HTML* é a linguagem de marcação para definir a estrutura das páginas estaticamente, enquanto o *JavaScript* é a linguagem de programação projetada primariamente para fazer tarefas dinâmicas com tais páginas *HTML* (CHAPMAN, 2018), como por exemplo atualizar dinamicamente páginas *Web*, oferecer melhorias na interface do usuário, animações, gráficos 2D e 3D, mapas interativos, *players* de vídeo, entre outros (SRIDHAR, 2017). É uma linguagem suportada pela maioria dos *browsers Web*, como o *Chrome, Firefox, Internet Explorer, Edge, Opera, Safari*, entre outros, incluindo *browsers* para *smartphones* (SRIDHAR, 2017).

Por essa e outras razões, o *JavaScript* é uma linguagem largamente utilizada, conforme podemos observar nas figuras 2, 3, 4 e 5. Apesar de hoje também ser possível utilizá-la como *server-side*, ela foi utilizada apenas como uma linguagem *client-side* neste projeto (CHAPMAN, 2018), sendo o *PHP* utilizado como linguagem padrão pra programação do *Web Service*, enquanto o *JavaScript* busca os dados dinamicamente por *AJAX*.

3.5.1 AJAX

AJAX (*Asynchronous JavaScript and XML*) é uma técnica de desenvolvimento *Web* que permite a troca de informações com um servidor *Web* nos bastidores, isto é, sem parar a execução ou atualizar a página (DEVMEDIA, 2007).

É interessante o que (W3SCHOOLS, 2018a) diz sobre *AJAX*, o colocando como o sonho do desenvolvedor, pois com ele é possível ler os dados de um *Web Server* depois que a página é carregada, atualizar a página sem precisar recarregá-la e enviar dados para um *Web Server* no *background*, enquanto o usuário utiliza a página. O *AJAX* usa uma combinação de um objeto *XMLHttpRequest*, que é interno dos navegadores, *JavaScript* e *HTML DOM*, que é um documento *HTML* carregado em um navegador *Web* (W3SCHOOLS, 2018a), permitindo que páginas *Web* sejam atualizadas assincronamente através de troca de dados com um servidor de forma transparente ao usuário, o que significa que é possível atualizar partes de uma página *Web* sem recarregar a página inteira (W3SCHOOLS, 2018a).

O *AJAX*, portanto, não é uma linguagem de programação, mas utiliza o *JavaScript* para fazer a troca de dados entre o servidor e a página *Web*. Neste projeto foi utilizado *AJAX* para ser feita a troca de informações entre a aplicação do usuário e o *Web Service*. Para facilitar a utilização do *AJAX* e outras funções úteis do JavaScript no projeto, foi utilizado a biblioteca *jQuery*.

3.5.2 jQuery

Segundo o site oficial, o *jQuery* é uma biblioteca *JavaScript* rápida, pequena e com riqueza de *features*, tornando mais simples a manipulação de documentos *HTML*, *event handling*, animação e requisições *AJAX* com uma *API* fácil de utilizar que funciona através de diversos *browsers* (JQUERY, 2018).

O propósito da utilização do *jQuery* é tornar mais fácil o uso do *JavaScript* nos *sites Web*, pegando várias tarefas comuns que requerem diversas linhas de código pra serem feitas e as juntam em métodos que você pode chamar com uma única linha de código, simplificando tarefas mais complexas como chamadas *AJAX*, manipulação do *DOM* (*Document Object Model*), manipulação do *CSS*, métodos de eventos *HTML*, efeitos, animações, entre outras utilidades (W3SCHOOLS, 2018e).

Portanto, foi utilizada a biblioteca *jQuery* para manipulação de objetos na tela da aplicação, além de uso extensivo de requisições *AJAX* para consumir os serviços do *Web Service*.

3.6 MYSQL

Para prestar os serviços, o *Web Service* se utiliza de um banco de dados para persistir alguns dados, como por exemplo exercícios pré-cadastrados em listas, para que seja possível ao usuário receber exercícios conforme a lista escolhida e o tipo de exercício buscado (tableaux, resolução, semântica, entre outros). Além disso, o usuário pode acrescentar exercícios próprios, que deverão ser persistidos no banco de dados.

Foi utilizado para esses casos um banco de dados relacional *MySQL*, que é rápido, confiável e fácil de configurar e utilizar, sendo gratuito o uso e o sistema de banco de dados mais popular usado com *PHP* (W3SCHOOLS, 2018f). Isso faz com que a configuração do *Laravel* com o sistema de gerenciamento de banco de dados seja feita de forma rápida e direta, com suporte nativo para que o *Web Service* consiga proporcionar todos os serviços de persistência necessários.

Por fim, é importante frisar que toda a parte de segurança, tanto do sistema de gerenciamento de banco de dados quanto do servidor é feito pelo *Laravel*, sendo transparente para o desenvolvedor.

4 IMPLEMENTAÇÃO

Neste capítulo serão discutidas todos os pontos que foram programados no sistema, desde a modelagem até as funções que foram programadas propriamente ditas, destacando apenas as funções essenciais e a ideia do sistema para não tornar o texto desnecessariamente prolixo.

4.1 MODELAGEM DO BANCO DE DADOS

Conforme mencionado na seção 3.6, o *Web Service* utiliza um banco de dados *MySQL* para persistir dados como categorias, listas de exercício e exercícios. A figura 8 mostra o Modelo Entidade Relacionamento do banco de dados do projeto.

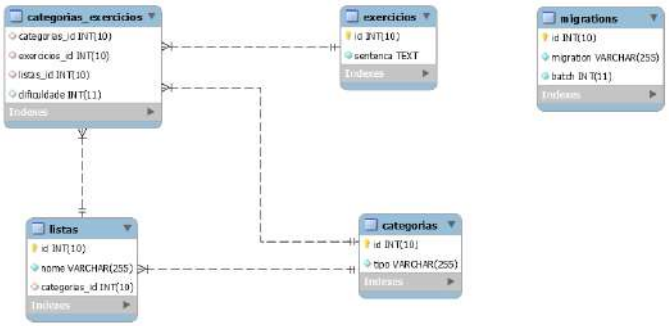


Figura 8 – Modelo ER do banco de dados

É importante destacar que a tabela *migrations* é uma tabela criada pelo *framework Laravel* para gerenciar as operações com o banco que são realizadas, sendo uma tabela transparente ao desenvolvedor.

A tabela categorias é a tabela responsável pelo cadastro do tipo de exercício que uma lista ou um exercício é (tableaux, dedução natural, resolução, entre outros). Neste caso tanto a lista como exercícios separados podem ter categorias, já que nem todos os exercícios obrigatoriamente fazem parte de uma lista.

Uma lista é de apenas uma categoria, mas podemos ter várias listas de uma mesma categoria. Já um exercício pode ser resolvido por diferentes métodos, então ele pode ter várias categorias. Além disso, uma categoria tem vários exercícios, o que explica a relação N pra N neste caso.

4.2 WEB SERVICE

Como explicado no capítulo 3, o *Web Service* foi implementado em *PHP* utilizando o *framework Laravel*, que por sua vez utiliza o gerenciador de dependências *Composer* (ver seção 3.2.1).

Para a correta instalação e configuração do Laravel e do projeto localmente, é necessário seguir os passos descritos a seguir.

1. Entre no site do *Composer* (https://getcomposer.org) e instale-o (Ver no site a documentação para instalação do *Composer*)
2. Abra um novo terminal ou prompt de comando se no *Windows*
3. No terminal, digite: ***composer global require "laravel/installer"***
4. Ainda com o terminal aberto, navegue até a pasta ***LogicAPI*** dentro da pasta do projeto e rode o comando: ***composer install***
5. Ainda com o terminal na pasta, rode o comando: ***php artisan key:generate***
6. No terminal, rode o servidor *PHP* executando o comando: ***php artisan serve***, e teste a aplicação no navegador utilizando o endereço provido pelo comando. Uma página com o escrito "WebService Funcionando!"deve aparecer.
7. Instale o *MySQL* e opcionalmente suas ferramentas
8. Crie uma nova conexão em Localhost (cada ferramenta possui uma forma diferente de criar conexão, veja o manual de utilização do *MySQL* e da ferramenta utilizada para mais informações)
9. Crie um *Schema* com nome ***logicapi***
10. Marque o *Schema* previamente criado como padrão
11. Volte para o terminal e certifique-se que está na pasta ***LogicAPI***
12. Rode o comando: ***php artisan migrate***, para criar as tabelas do banco de dados
13. Rode o comando: ***php artisan db:seed***, para popular o banco de dados com os dados padrões (exercícios, listas e categorias).
14. Verifique se todos os dados estão no banco. O *Web Service* está pronto para ser utilizado

4.3 MÉTODOS IMPLEMENTADOS

Nesta seção discutiremos a implementação propriamente dita. Como foi utilizada a linguagem *PHP* neste projeto, por conveniência maior parte das estruturas de dados são arrays e hashs, de modo que por indexação se pode acessar diretamente os elementos. Foi-se estabelecido um padrão comportamental para cada método implementado afim de definir a estrutura básica dos arrays a serem utilizados, bem como todas as mudanças que as fórmulas de Lógica poderiam sofrer, afim de garantir uma fácil manipulação de baixo custo de processamento. Essencialmente a codificação dos métodos abaixo é composta de duas funções que se comunicam com o Controller da aplicação, a primeiro delas é a função ***Full Steps***, esta função recebe a lista de fórmulas do Controller e aplica todas as operações da forma como o pseudocódigo do método em questão descreve, gerando um resultado ao final e retornando ao Controller da aplicação, fazendo portando apenas duas vezes esta comunicação, esta função serve para realizar a requisição gerada quando o usuário clica na opção **Gabarito** descrita no *frontend* na seção. A segunda é a função ***step-by-Step***, esta função, diferentemente da primeira, se comunica várias vezes com o Controller, dependendo das requisições que forem geradas no *frontend*, como o *backend* do webservice não mantém o estado das variáveis, as fórmulas precisam ser passadas todas as vezes, juntamente com a operação que se quer fazer sobre elas. Esta função funciona como caso default em termos do *frontend* pois, este, lista todas as operações possíveis de modo que seja intuitivo para o usuário ir interagindo com elas.

4.3.1 Resolução

O método de Resolução que só fora apresentado no curso no contexto da Linguagem Lógica Proposicional possui um algoritmo bem simples, a principal complexidade deste método se dá nas várias possibilidades de manipulação de fórmulas. Segue abaixo a ideia geral por trás do algoritmo da resolução.

Algoritmo 3 Algoritmo base de resolução

Entrada: Uma lista de fórmulas, inclui a pergunta
Saída: Um conjunto de fórmulas após a aplicação de cada regra e o status fechado ou não-fechado ao final das aplicações de regras.

1. Ler todas as fórmulas armazenando-as numa lista
2. Inicialziar status "não-fechado"
3. Negar a pergunta a pergunta
4. Passar todas as fórmulas para FNC (conectivo e não pode estar dentro de parênteses)
5. Repita até ser respondido status fechado ou acabarem as possibilidades
6. Separar todos os conectivos "e"de modo que cada subfórmula seja uma fórmula independente
7. Listar todos os átomos que foram gerados (caso haja algum)
8. Se houver átomos de com valor oposto responda status "fechado"
9. Se não houver átomos com valor oposto continua
10. Para cada par de fórmulas disponível fazer as simplificações com o conectivo "ou"(Se $A \vee \neg B$ e $A \vee B$ então A. Se $A \vee \neg B$ e B então A).
11. Se status "não-fechado"então a resolução não fecha.
12. Responda o status

Este método foi escrito em uma classe própria chamada e possui as duas funções, fullsteps e *step-by-Step*. Também possui uma outra classe chamada *FuncoesResolucao* que tem como objetivo conter todas as operações de manipulação das fórmulas, para simplificar o código da classe *Resolucao* de modo a torná-lo parecido com o algoritmo supracitado

4.3.1.1 Estrutura de Dados

Convém definir a estrutura de dados para simplificar a explicação do método logo abaixo, assim, no contexto de resolução chamaremos de ***fórmula*** um array com 3 campos:

- **Esquerdo** - Guarda uma string ou um array fórmula com a subfórmula da esquerda
- **Conectivo** - Guarda um caracter com o conectivo de maior prioridade da fórmula, o conectivo no qual seria aplicada uma operação.
- **Direito** - Guarda uma string ou um array fórmula com a subfórmula da direita

Sobre o fechamento da resolução

Para sabermos se os átomos "casam", ou seja, se há dois átomos de valores opostos, foi criada uma hash para armazenar todos os átomos que forem sendo gerados a medida em que se simplificam as fórmulas. Atribuímos como chaves da hash o átomo e como valor "0"para átomo negativo ou "1"para átomo positivo, de modo que se já existe por exemplo um átomo de chave A e valor 0 na hash, caso a aplicação de alguma regra resulte num novo A com valor 1 sabemos que foi encontrada uma contradição e portanto a resolução fecha.

4.3.1.2 Full Steps

Dado que este método segue essencialmente o algoritmo supracitado, serão destacados alguns pontos chaves que foram importantes para resolver problemas. Existem essencialmente 6 operações implementadas para realizar todos os passos do algoritmo da resolução, são elas:

- *Negação da pergunta* - Recebe o array das fórmulas e retorna com o último elemento do array negado, se o problema possui apenas a pergunta, então esse único elemento do array é negado.
- *Remoção de NOTNOT* - Recebe todas as fórmulas e faz um pré-processamento de onde sequências de negação $\neg\neg$ são removidas, o objetivo é se aproveitar do fato de que essa sequência de NOTs não interfere no valor da fórmula para evitar complicações na manipulação de fórmulas que gerem mais NOT sobrepondo esses anteriores.
- *Fórmula em FNC* - Recebe todas as fórmulas e as coloca na Forma Normal Conjuntiva de acordo com o [algo2.4.3.1]**primeiro algoritmo em 2.4.3**
- *Separação do E* - Recebe todas as fórmulas e separa os conectivos E mais externos existentes gerando para cada fórmula, duas novas fórmulas.
- *Separação do OU* - Recebe todas as fórmulas e para cada par de fórmulas verifica se é possível fazer uma simplificação aproveitando-se das propriedades do conectivo OU, caso seja possível, gerará uma fórmula derivada de uma das duas no par sem o conectivo OU mais externo.
- *Move NOT para dentro* - Recebe todas as fórmulas e verifica fórmulas em que haja um conectivo $\neg$ fora de parênteses, havendo esta operação aplicará o $\neg$ nas subfórmulas e converterá o conectivo mais externo ($\wedge$ passa a ser $\vee$ e $\vee$ passa a ser $\wedge$).

Para facilitar a manipulação do *frontend* sobre os resultados gerados, existe um vetor *resposta* que recebe várias 3-upla de elementos/conjunto de elementos, seguindo o formato *{Operação, array de fórmulas antes da aplicação, array de fórmulas depois da aplicacão}*, levando em conta que a *"Operação"* será uma das 6 supracitadas, deste modo quando o *frontend* lê o vetor resposta ele sabe exatamente como mostrar os resultados considerando que a operação é sempre lida primeiro. Ao final da aplicação do algoritmo é retornado o par com vetor *resposta* e o *status* "fechado"ou "não-fechado"que é encaminhado para o Controller da aplicação, este por sua vez se encarrega de encaminhar para o *frontend* na rota correspondente como resposta ao se clicar no botão **Gabarito**.

4.3.1.3 *step-by-Step*

Esta função funciona em sua essência da mesma maneira que a função *full-steps*, mas diferentemente de *full-steps* que só dá um retorno após ser acionada a opção de Gabarito, esta retorna várias vezes de acordo com as manipulações que o usuário desejar fazer nas fórmulas. As operações que antes apenas eram retornadas para permitir a organização do *frontend*, agora são passadas do *frontend* para o *backend* da aplicação, como critério para ativar o passo do código correspondente, uma vez que o *backend* não sabe o que está sendo feito no *frontend*. Assim, em cada iteração, o *backend* recebe o nome da operação que se deseja aplicar e um conjunto de fórmulas onde se deseja aplicar a operação especificada, retornando apenas o resultado que pode ser uma ou mais fórmulas.

4.3.2 Tableaux

O método do Tableaux por sua vez é um pouco mais complexo do que a resolução a implementação requereu muitas considerações para garantir a integridade das operações aplicadas sobre a fórmula. Formulou-se um algoritmo base de acordo com o método do Tableaux ensinado no curso de lógica , segue abaixo:

Algoritmo 4 Algoritmo base de tableaux

Entrada: Uma lista de fórmulas, inclui a pergunta
Saída: Uma árvore onde cada nó folha pode ter o status "fechado"ou "não-fechado".

1. Ler todas as fórmulas armazenando-as numa lista
2. Negar a pergunta a pergunta
3. Para cada fórmula do banco de dados ou a pergunta ou das fórmulas não atômicas que são nós folha faça:
4. Escolha uma fórmula
5. Aplique a regra correspondente ao conectivo mais externo
6. Aponte para os filhos da fórmula, respeitando caso o conectivo gere 1 ramo ou 2 ramos
7. Verificar para todos os ramos gerados nesta etapa se fecham
8. Se todos os ramos estiverem fechados PARE

Este método, diferentemente da resolução possui versões da Lógica Proposicional e de Primeira Ordem, de modo que tivemos de criar uma classe para cada afim de permitir um desenvolvimento em paralelo e evitar propagação de erros com as peculiaridades de manipulação da estrutura de dados de cada uma.

4.3.2.1 Versão da Lógica Proposicional

Estrutura de dados

Para facilitar o refenciamento desta estrutura será denominado que o array específico da classe Tableaux será **fórmula tableaux** Segue abaixo a lista de campos que compõem o array de uma fórmula tableaux.

- **info** - Este campo contem as informações que são de fato da fórmula, composto por um array de 3 campos: Esquerdo, concectivo e direito.
- **atualEsquerdo** - Campo que informa se esta fórmula é filho da esquerda do seu pai.
- **atualDireito** - Campo que informa se esta fórmula é filho da direita do seu pai.
- **atualCentral** - Campo que informa se esta fórmula é filho central do seu pai.
- **filhoEsquerdo** - Campo que contem um array *fórmula tableaux* do filho da esquerda deste nó.
- **filhoCentral** - Campo que contem um array *fórmula tableaux* do central deste nó.
- **filhoDireito** - Campo que contem um array *fórmula tableaux* do filho da direita deste nó.
- **pai** - Campo que contem um array *fórmula tableaux* do pai deste nó.

- **formDisponiveis** - Campo que contem a lista de todas as fórmulas em que ainda se podem aplicar regras que são antecessoras á este nó.
- **hashAtomos** - Campo que contem uma hash com chave o caractere do átomo e valor 0 para negativo ou 1 para positivo.

As estruturas de dados do tableaux são mais complexas que da resolução e foram inicialmente inspiradas na estrutura de árvore binária (TUTORIALSPOINT, 2018) e devidamente modificadas para poder-se trabalhar com a flexibilidade requerida pelo tableaux, possuindo o campo pai, por exemplo. A mudança mais facilmente percebida são 3 campos filhos. Na tradicional árvore binária se tem um campo para o *Filho da esquerda* e outro para o *Filho da direita*, aqui acrescenta-se o campo *Filho Central* de modo que este campo pode ou não aparecer, a depender se a aplicação da regra gera apenas um filho (este será o filho central) ou se gera dois (esquerda e direita).
Os campos *atual direito, esquerdo e central* não tem uso durante o processamento da árvore em si, porém foram concebidos para permitir maior controle na hora da impressão garantido uma visualização mais perfeita pelo *frontend*.
O campo *formDisponiveis* foi concebido para uma lógica específica do tableaux, é necessário saber dentre as fórmulas em que ainda se pode aplicar regras, qual escolher, sendo que estas fórmulas mudam de acordo com o nó em que se está. Considere por exemplo, o caso em que se desce vários níveis para esquerda e outro em que se desce vários níveis para direita. A fórmula folha da esquerda não terá necessariamente as mesmas fórmulas disponíveis que a da direita, tendo isto em vista, foi necessario agregar essa informação para cada nó da árvore.
Por fim, o campo *hashAtomos* é importante para o fechamento de um ramo, com esta informação agregada para cada nó, sabe-se exatamente todos os nós atômicos ascendentes dele, de modo que se um átomo de valor oposto for gerado na aplicação da regra no nó atual, se saberá de imediato que o ramo fechou.

Função *full-steps*

Com esta formulação e estruturas a função *full-steps* da classe Tableaux possui poucas linhas de código, meramente executando em loop a chamada da função que escolhe qual o nó o qual deve ser aplicada uma regra. Fazendo a escolha de um modo razoavelmente eficiente em que se começa sempre pelas regras que geram apenas um filho/ramo para diminuir a complexidade da árvore final gerada. Uma vez que a árvore é gerada pelo sistema de ponteiros, ela é passada para o *frontend* utilizando o método de impressão *pré-ordem*, para mostrar a árvore de um jeito bonito e legível para o usuário.

4.3.2.2 Versão da Lógica de Primeira Ordem

A ideia base e o algoritmo seguidos são essencialmente os mesmos da versão da lógica proposicional, mas foi criada a *Classe TableauxLPO* para se explorar as particularidades da *Classe Tableaux* sem comprometer o código da anterior, usando as mesmas funções quando possível.

Estrutura de Dados

Na versão proposicional usamos uma fórmula contendo os campos, *esquerdo, conectivo e direito*, porém, aqui as os átomos podem ser funções e precisa-se saber qual é a variável referenciada por cada quantificador de modo que a fórmula agora o conectivo passa a ser um array com dois campos, sendo eles o campo **operacao** que contem o tipo de conectivo propriamente dito ($\exists$, $\wedge$, $\vee$, $\rightarrow$ $\forall$, $\neg$) e o campo **variavel** que contem a variável que referenciada caso a operação seja um quantificador, ou **null** caso contrário.

O próximo campo novo é **hashAtomosFuncoes** que é uma hash específica para funções cuja a variável já foi substituída por uma constante, funcionando do mesmo jeito que *hashAtomos* do tableaux proposicional, onde se possui uma função como chave e o valor 0 para negativo ou 1 para positivo.

A seguir temos o campo **constantesUsadas** que é um array contendo todas as constantes que já foram usadas naquele ramo, este campo foi a **solução** encontrada para manipular facilmente fórmulas em que a constante só pode aparecer uma vez no ramo, assim, com uma consulta direta das constantes das fórmulas ascendentes é tomada a decisão sobre qual constante utilizar na fórmula atual.

Por fim, o último campo novo é o campo **funcoesComSubstituição** que foi concebido para buscar-se facilmente se já existe uma mesma função como fórmula ascendente para que se escolha diretamente uma constante que case com esta função ascendente para garantir o fechamento do ramo.

Função *full-steps*

A função *full-steps* do tableaux de lógica de primeira ordem funciona da mesma maneira que a função do tableaux da lógica proposicional, porém aqui enfrentou-se uma dificuldade de fazer a escolha das constantes corretas na hora de aplicar as regras quantificadoras afim de garantir o fechamento das árvores de tableaux. Esta decisão é tomada na função de aplicação de regra contida no *full-steps*, que é chamada da mesma forma que no Tableaux proposicional, porém com a implementação ligeiramente diferente que explora os novos campos da estrutura de dados supracitados para tomar a decisão da melhor maneira possível.

4.3.3 Dedução Natural

O método de dedução natural tem uma lógica de implementação um pouco diferente dos outros métodos do projeto. Este é o único método em que o provador automático nao foi implementado, apenas verificando passo a passo as regras aplicadas pelo usuário à questão e retornando o resultado final da regra caso ela possa ser aplicada para as fórmulas selecionadas.

4.3.3.1 Versão da Lógica Proposicional

Para este trabalho apenas a versão para Lógica Proposicional foi implementada. A aplicação do usuário é responsável por avaliar se a resposta gerada pelo *Web Service* é igual a resposta esperada pela pergunta do exercício, sendo portanto o responsável por dar o exercício como finalizado.

Algoritmo 5 Algoritmo base da dedução natural

Entrada: Uma *string* informando a regra que se deseja aplicar, uma lista com todas as fórmulas selecionadas e/ou uma *string* contendo uma fórmula que se deseja incluir ou supor

Saída: Fórmula resultante da aplicação da regra ou mensagem de erro ao usuário caso seja detectado um fator de impedimento em uma das etapas de avaliação e verificação

1. Verificar a regra que se deseja aplicar
2. Avaliar se todos os parâmetros necessários para a aplicação da regra foram passados
3. Avaliar se foram selecionados a quantidade de fórmulas corretas para aplicação da regra
4. Verificar se as fórmulas selecionadas fazem parte do mesmo escopo
5. Avaliar se a regra selecionada pode ser aplicada para a(s) fórmula(s) selecionada(s)
6. Aplicar a regra

4.3.4 Semântica

O método da semântica foi razoavelmente simples de implementar, mas requereu algum conhecimento específico de estrutura de dados para poder-se manipular a árvore com liberdade. Seguindo a explicação dos métodos anteriores, será formulado um pseudocódigo de implementação que e explica como o método da Semântica pode ser feito. O pseudocódigo também pode ser usado por alguém que queira fazer a Semântica "na mão". O algoritmo acima gera a árvore de acordo com as decisões do usuário permitindo facilmente analisar interpretações que satisfaçam ou não a fórmula de entrada.

4.3.4.1 Estrutura de dados

A estrutura de dados da semantica requereu um conhecimento não estudado previamente que está no contexto de *Árvores com número de filhos variável*, para tal usou se a

Algoritmo 6 Algoritmo base de semantica

Entrada: Uma pergunta

Saída: Uma árvore que desmembra a fórmula e uma interpretação sobre esta árvore.

1. Ler a fórmula e armazená-la.
2. Defina um array de domínio para a análise dessa fórmula.
3. Enquanto houver conectivo na fórmula faça:
4. Se o conectivo mais externo for um quantificador, gere um ramo para cada valor do quantificador substituindo na variável correspondente na fórmula.
5. Senão se o conectivo mais externo não for um quantificador, gere dois ramos separando os dois lados do conectivo.
6. Para cada função que compõe a fórmula escolha um array de relações para esta fórmula
7. Para cada relação definida, faça a validação da fórmula de baixo para cima.
8. A interpretação dependente das relações geradas acima será o resultado final da fórmula.

ideia de (CELES; RANGEL, 2002) na seção 13.2 *Árvores Genéricas* e adaptou-se o conhecimento de subárvores para permitir projetar uma árvore de busca em largura Segue abaixo a lista de campos:

- **info** - Contém um array de fórmula com os mesmos campos de 4.3.2.2 parágrafo 1 de Estrutura de Dados
- **filhos** - Contém um array com o número de filhos deste nó
- **pai** - Contém uma referência par ao pai deste nó.
- **valor** - Contém um booleano que representa o valor lógico verdadeiro ou falso
- **usado** - Contém um booleano onde true significa que este nó já foi percorrido e atribuido seu valor lógico ou false para não percorrido.
- **proximo** - Contém a referência para o irmão a direita deste nó

Destaca-se que o campo **valor** é inicializado como falso pois os nós não contidos na relação serão falsos e a árvore é percorrida duas vezes, uma no processamento e outra na impressão, garantido a integridade da estrutura e facilitando correções de bugs desse modo coonsiderando que campo **usado** foi criado para garantir que todos os nós foram percorridos.

4.3.4.2 Full Steps

Dado que este método requer pouca iteração do usuário apenas o método *full-steps* foi implementado, recebendo as entradas do usuário e fazendo todo o trabalho braçal necessário para analisa-la e tomar decisões sobre as interpretações da árvore. O algoritmo

é implementado de acordo com o pseudocódigo acima, porém fazendo-se uma busca em largura na geração da árvore, diferentemente do Tableaux que trabalha com busca em profundidade com backtracking. Com isto o código da Semantica foi o menor e mais elegante implementado de modo que a classe Semantica possui menos de 50 linhas e utiliza funções da classe FunçõesSemantica para fazer os processamentos necessários e secundários

4.4 INTERFACE GRÁFICA

O objetivo básico da interface gráfica é deixar a utilização do sistema bem intuitiva. Inicialmente, o usuário encontrará uma aba onde ele deverá escolher o tipo de lógica que gostaria de praticar, podendo ser proposicional ou primeira ordem. A primeira interface com que o usuário se depara conta com elementos básicos de seleção e navegação, como tabs, ratio-buttons e um botão simples para a troca de aba.

Figura 9 – Primeira tela

Nesta mesma aba, após o usuário selecionar o tipo de lógica a ser utilizada, deverá ser escolhido o tipo de método a ser utilizado ou alterar o tipo de lógica selecionado anteriormente.

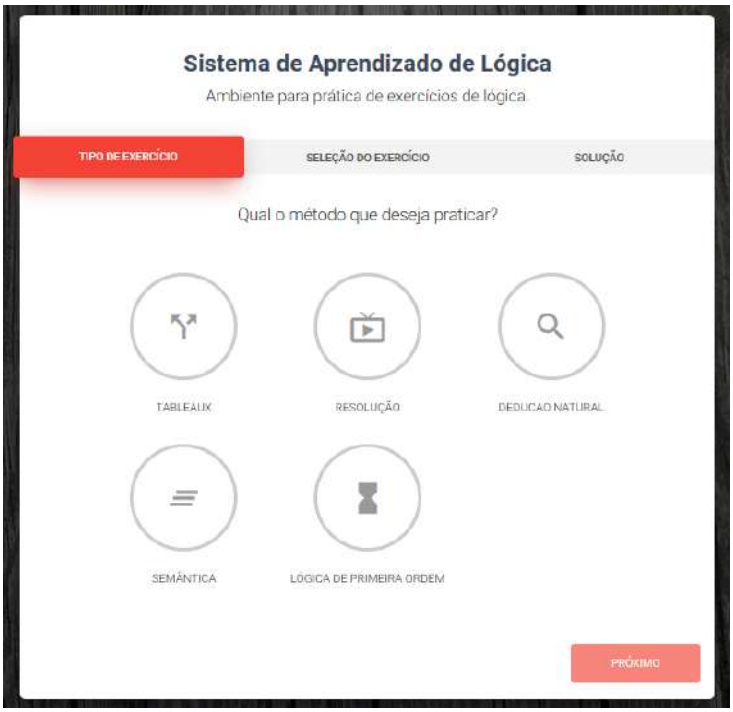


Figura 10 – Métodos da lógica proposicional

Ao ser escolhido um método a lista de exercícios correspondente é mostrada, seguem abaixos as telas com as listas disponíveis

Figura 11 – Métodos da lógica de primeira ordem

Figura 12 – Lista de exercícios de Lógica Proposicional

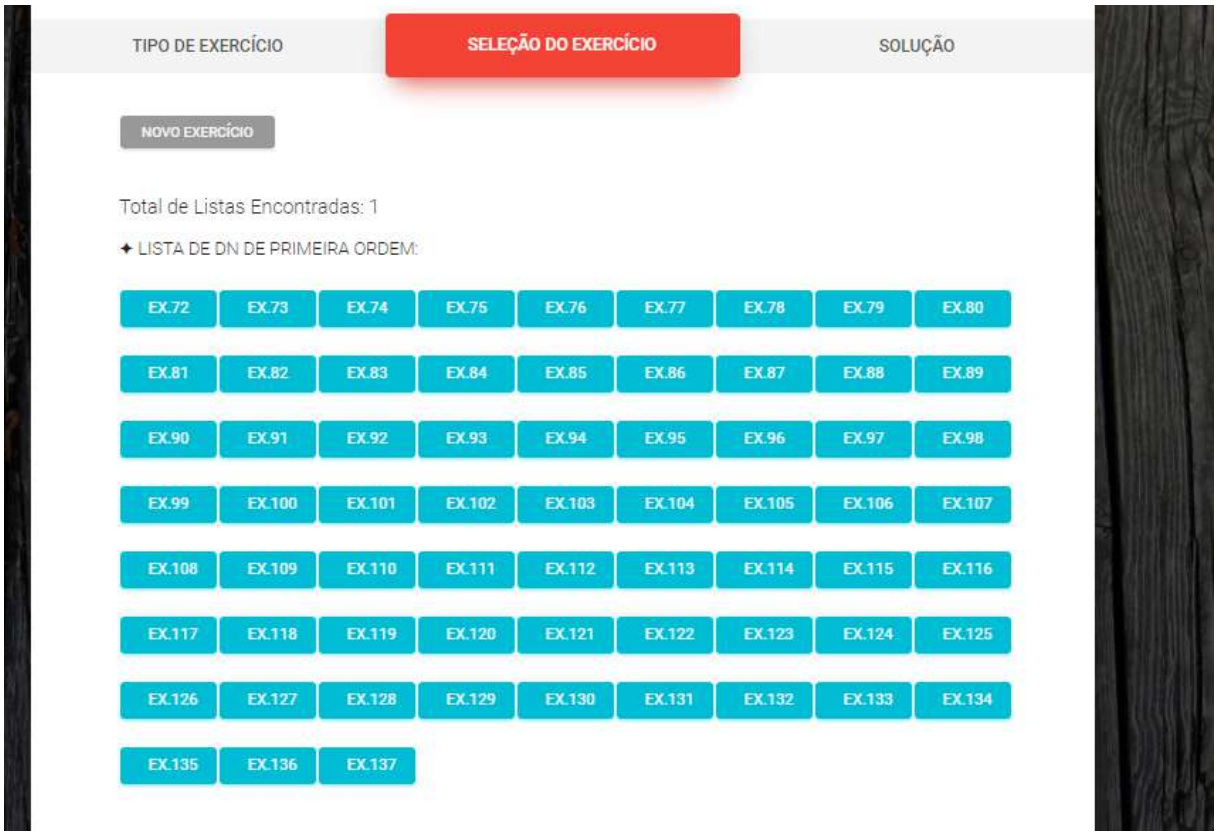


Figura 13 – Lista de exercícios de DN de Primeira Ordem

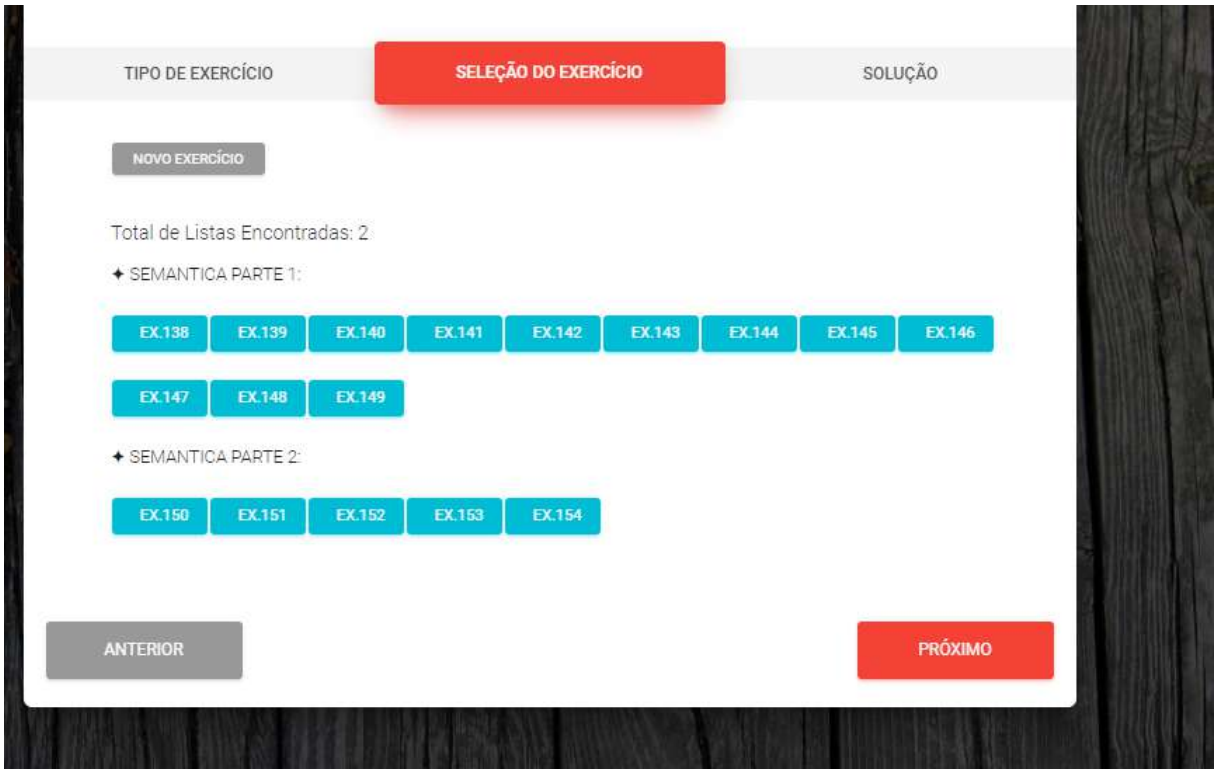


Figura 14 – Lista de exercícios de Semântica de Primeira Ordem

A cada seleção de método ou mudança do tipo de lógica é feita uma requisição ao backend para buscar as listas e os exercícios relativos a escolha do usuário. Com isso, para evitar bugs, a cada escolha feita e mudança de aba as variáveis que armazenam as regras dos exercícios e as divs com as informações dos exercícios precisam ser redefinidas. Por esse motivo, a função de limpar tudo que foi definido em múltiplas abas se torna uma das mais importantes do frontend.

Outro ponto importante implementado que merece um destaque maior é a validação do exercício digitado pelo usuário. Esta validação faz com que, antes mesmo do exercício começar a ser desenvolvido, o usuário tenha um feedback caso o problema não possua solução e também impede que o usuário entre com dados incorretos, como por exemplo um número ímpar de parênteses.

Para uma programação coesa, respeitando as boas práticas de programação, cada método possui um arquivo de extensão js individual. Isto facilitou as diversas alterações que foram sendo feitas ao longo do projeto, permitindo que mesmo quem não trabalhou diretamente no código do frontend pudesse compreender a estrutura do projeto e a codificação. Além disso, foi criado um arquivo com funções comuns que não variam de método para método, chamado "funções gerais.js". Este arquivo além de ficar responsável por funções como as citadas anteriormente que solicitam acesso ao banco para a busca de listas e exercícios, as funções que validam a entrada do usuário e também conter as variáveis

de armazenamento de fórmulas, ele funciona como um controller genérico, delegando as chamadas aos respectivos arquivos js de acordo com o cenário em que a plataforma de aprendizado se encontra, ou seja, de acordo com o conteúdo que o usuário está tentando aprender.

5 RESULTADOS

Neste capítulo serão apresentados os resultados efetivamente alcançados com o projeto, estabelecendo-se uma conexão direta com o capítulo 2 para mostrar como os métodos implementados estão utilizáveis no sistema.

5.1 MÉTODOS DE LÓGICA PROPOSICIONAL

5.1.1 Resolucao

O método da resolução é o método mais completo no sistema, pois possui as implementações *fullSteps* e *stepByStep* totalmente funcionais além de permitir que o usuário insira seus próprios exercícios para serem resolvidos, eleminando a dependência total com o banco de dados.
Ao escolher o método da resolução o usuário usuário será três possibilidades. A opção gabarito, resolver o exercício manualmente ou inserir o seu próprio. Abaixo segue um exemplo da interface após selecionar um exercício na resolução.

Figura 15 – Resolução - Exercicio 54

5.1.1.1 Gabarito

A função gabarito, conforme o nome sugere, consiste em basicamente resolver o exrcício de maneira automatica para o usuário, porém, apesar do nome gabarito, não existe uma

solução previamente cadastrada no sistema. Todos os cálculos são feitos em tempo de execução.

Para acessar esta opção, após o usuário selecionar o método da Resolução, basta selecionar um exercício da lista e clicar no botão gabarito, deste modo, um dos caminhos possíveis para se solucionar a questão, será apresentado ao usuário. Usuários que só queiram por exemplo conferir se uma resposta já feita possui um caminho válido, pode experenciar esta opção como um gabarito de fato, caso executem manualmente o **Algoritmo da Resolução** e com o auxílio do **Algoritmo de FNC** na conversão para FNC.

Segue abaixo um exemplo do funcionamento da função gabarito.

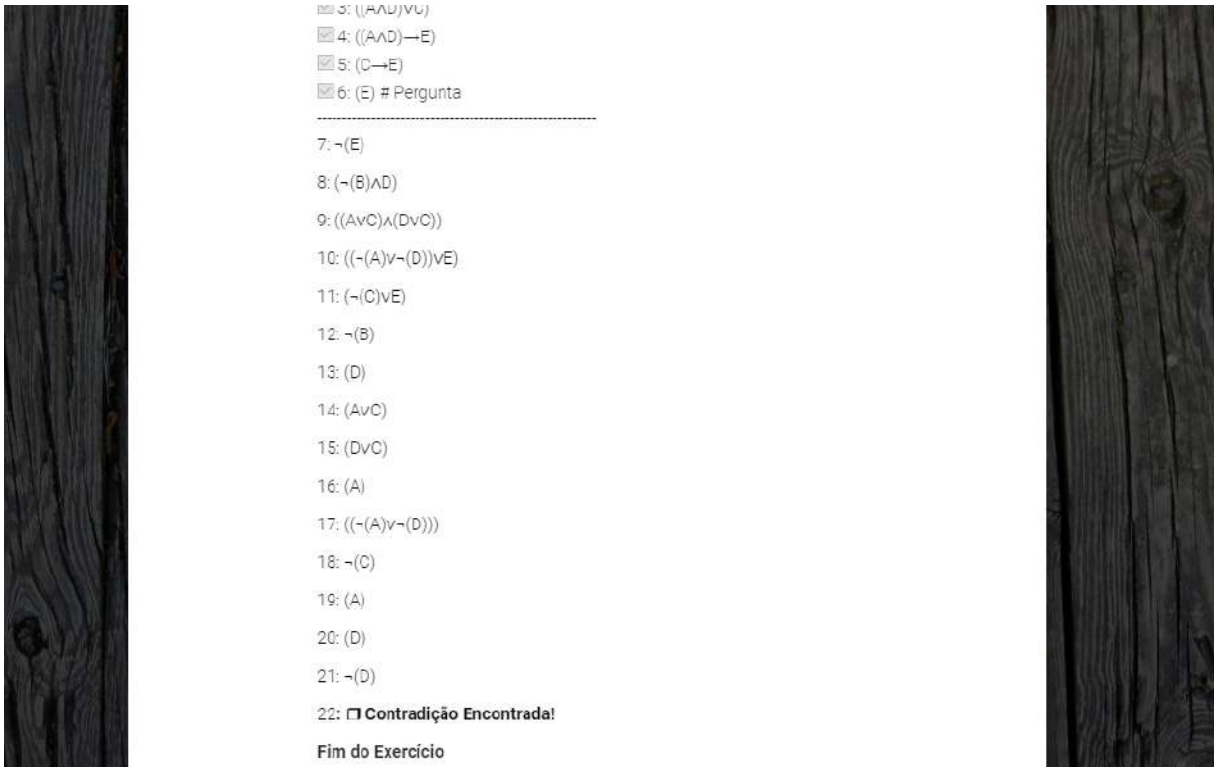


Figura 16 – Resolução - Função Gabarito do Exercicio 54

5.1.1.2 Solucionando exercício manualmente

Como o nome sugere, o usuário soluciona os exercícios por etapas manuais, porém, não é requisitada nenhuma entrada do usuário, ao invés disso, o usuário tem um leque de opções que podem ser realizadas sobre as fórmulas, deste modo o usuário não pode se permitir errar com erros simples, por exemplo, errar uma conversão de FNC e comprometer todo o restante do exercício. Basta que o usuário siga o algoritmo **Algoritmo de FNC** que conseguirá solucionar o exercício normalmente. Na tela, também são apresentadas algumas opções para orientar o usuário. Seguem abaixo imagem que exemplifica o funcionamento.

Figura 17 – Resolução - Passo a Passo do Exercicio 54

5.1.1.3 Solucionando exercício inserido manualmente

Esta possibilidade, embora de nome similar à seção anterior na verdade se refere ao fato de que o usuário pode inserir seus próprios exercicios de maneira completamente manual, eliminando portanto a dependência do banco de dados. Não é garantido que qualquer exercício possa ser solucionado, porém o sistema foi preparado para evitar que o usuário experencie qualquer erro do back-end nos caso algum exercício não seja possível resolver por falha na implementação ou porque o exercício realmente não possua solução lógica.

Da mesma forma o usuário deve-se atentar as instruções para garantir que exercícios funcionem corretamente e ao inserir um exercício manualmente ele poderá resolver tanto por gabarito como manalmente, seguem abaixo algumas imagems que exemplificam isto.

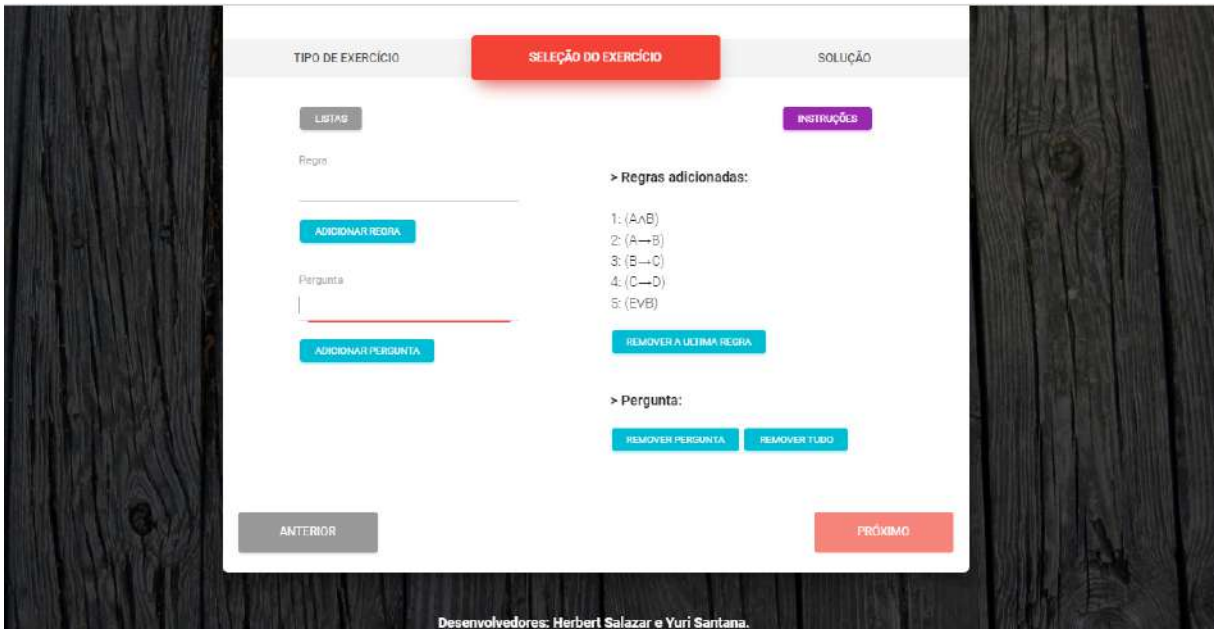


Figura 18 – Resolução - Inserindo exercicio manualmente

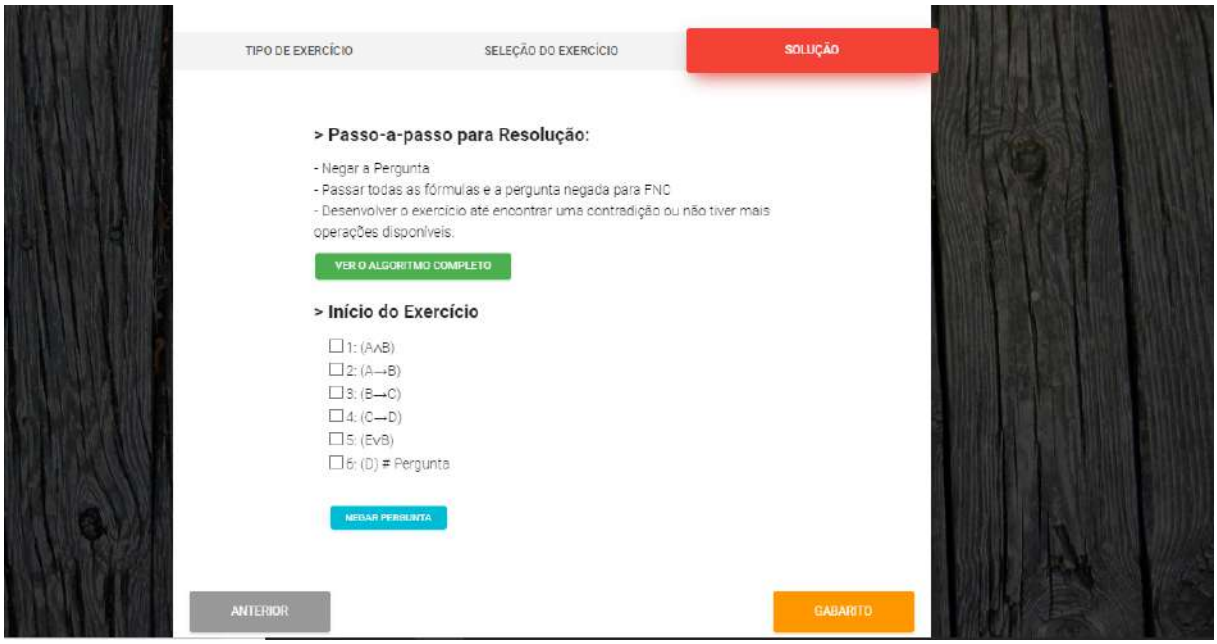


Figura 19 – Resolução - Exercício após inserção manual

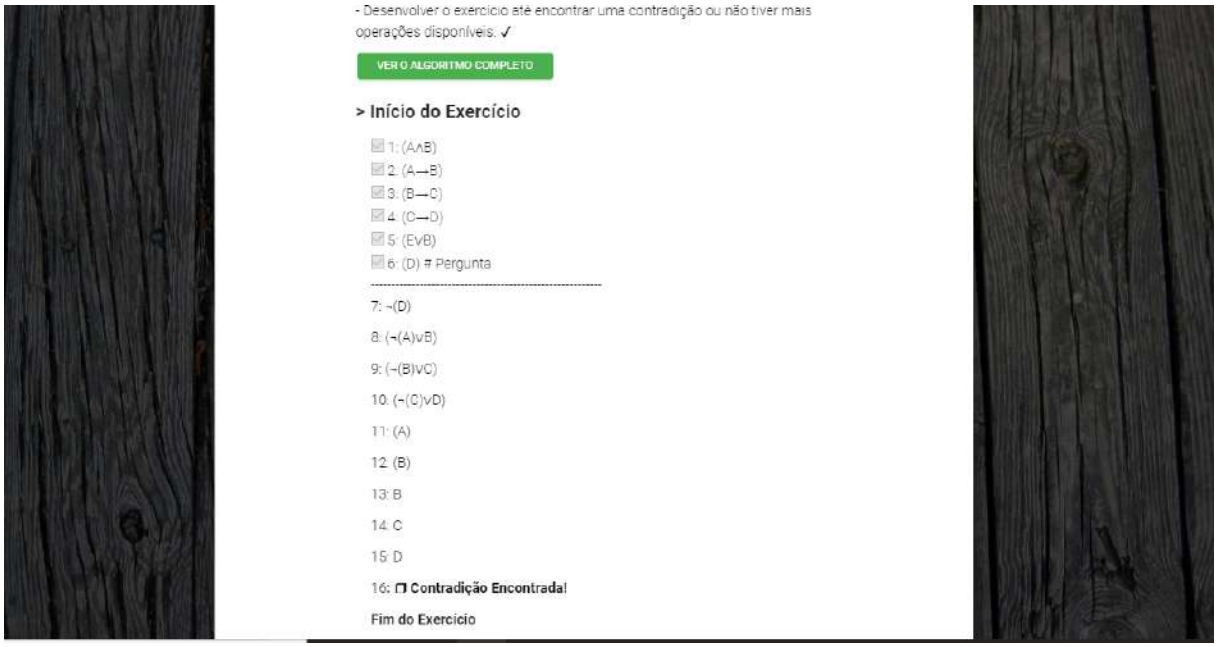


Figura 20 – Resolução - Função gabarito no exercício inserido manualmente

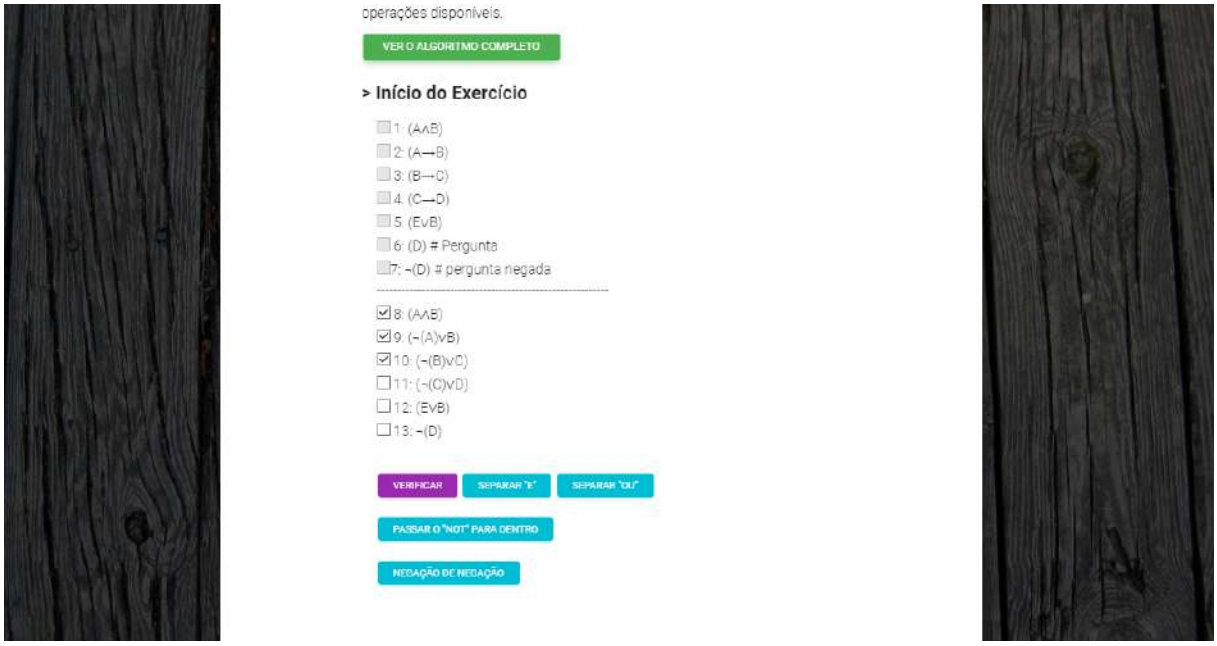


Figura 21 – Resolução - Passo a passo no exercício inserido manualmente

5.1.2 Tableaux

O método do Tableaux foi bem mais complicado de se implementar, tanto do ponto de vista do back-end quanto do front-end e deste modo, conseguiu-se preparar apenas a possibilidade de resolver exercícios via gabarito. Quando um ramo fecha, um nó é mos-

trado em vermelho, indicando que aquele ramo fechou, não significa que necessariamente aquele nó em vermelho foi responsável pelo fechamento do ramo. Seguem abaixo alguns exemplos de tableaux sendo solucionados.

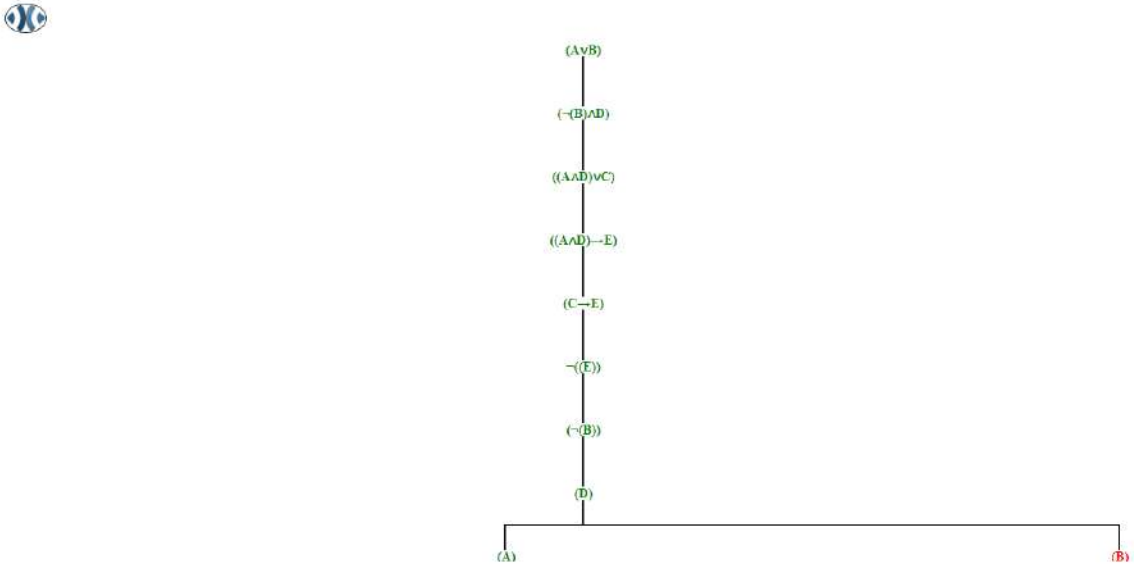


Figura 22 – Tableaux - Função Gabarito no exercício 54 - Parte 1

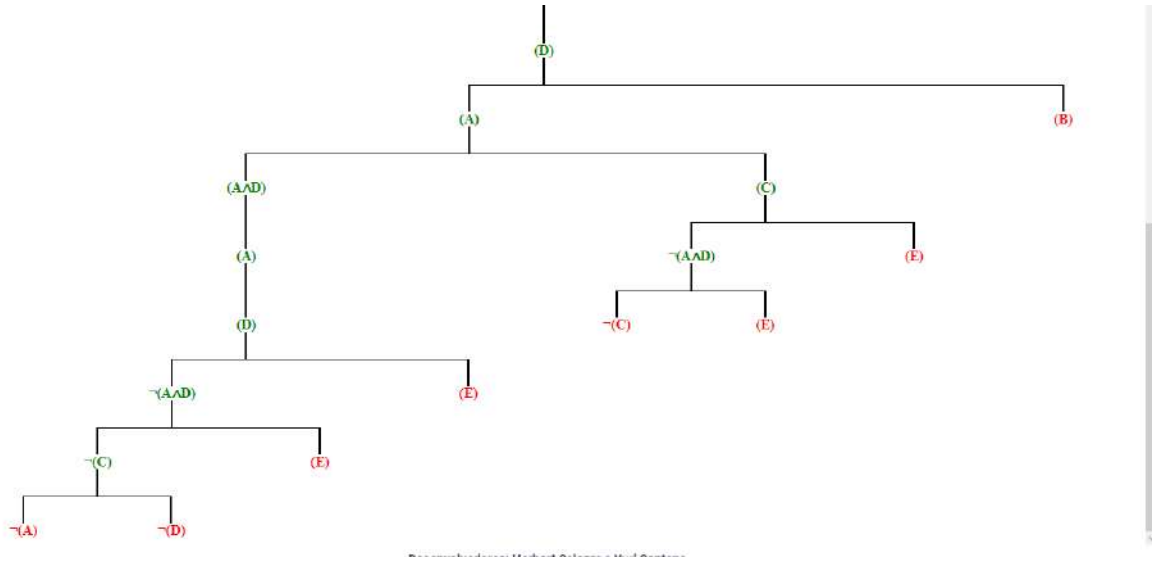


Figura 23 – Resolução - Função Gabarito no exercício 54 - Parte 2

5.1.3 Dedução Natural

O método da dedução natural não segue a mesma proposta dos outros métodos de mostrar um gabarito. Invés disso, ele permite que o usuário resolva fazendo passo a passo,

com o objetivo de garantir que o usuário não dê passos inconsistentes. Segue abaixo um exemplo de Dedução natural resolvida.

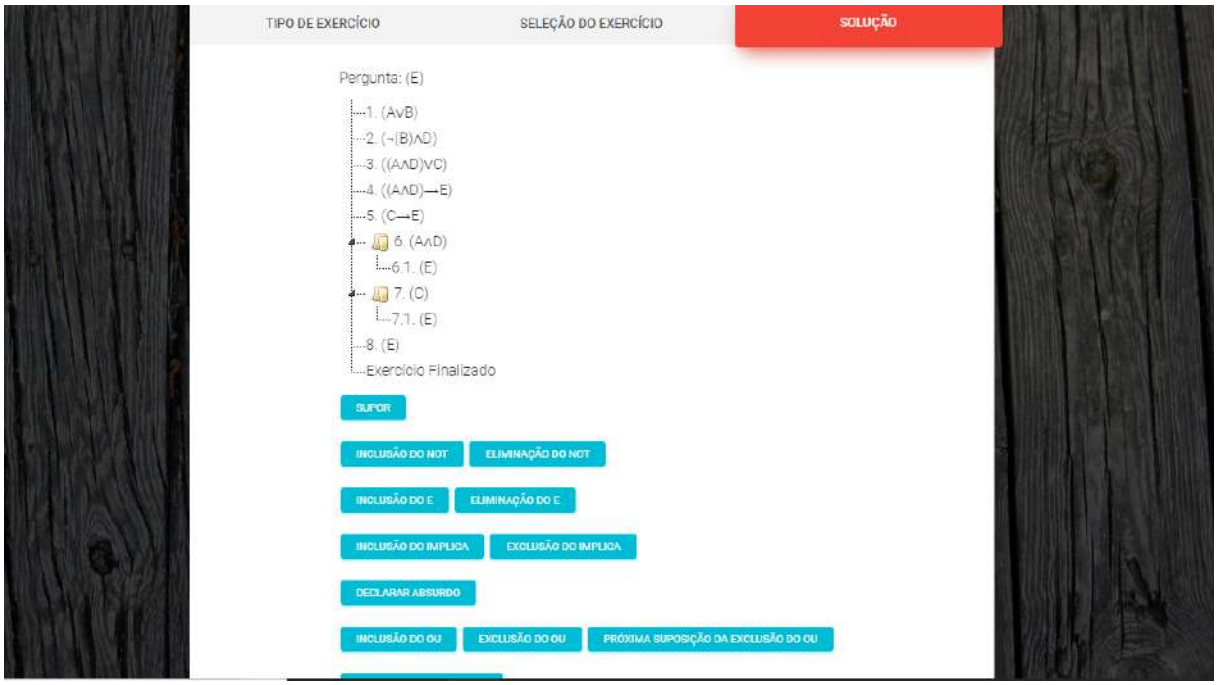


Figura 24 – Dedução natural - Exercício 54

No desenvolvimento da DN, o usuário tem a sua disposição todos os comandos disponíveis, sendo dois deles de entrada de texto livre. São estes a função "supor"e a "inclusão do Ou". Em ambos, o usuário tem a liberdade para inserir o texto que desejar de forma a ajudá-lo a chegar a uma solução, mas as mesmas regras de criação de exercícios devem ser respeitadas aqui, como por exemplo um número par de parênteses e a fórmula começando e terminando pelo mesmo. Caso o usuário tente executar uma função de forma incorreta, como por exemplo uma exclusão do implica onde não é possivel, um pop-up será exibido informando o erro.

5.2 MÉTODOS DE LÓGICA DE PRIMEIRA ORDEM

5.2.1 Tableaux

A mesma descrição da seção 5.1.2 se aplica aqui, destaca-se que o Tableaux de linguagem de primeira ordem teve uma implementação extremamente mais difícil que a do tableaux proposicional, pois o sistema deve tomar decisões sobre qual o melhor conectivo a ser selecionado para garantir que um ramo finalize. Outro ponto complexo foi também a conversão das estruturas de dados para String, afim de serem mostradas no front-end. Seguem abaixo um exemplo de tableaux sendo solucionado.

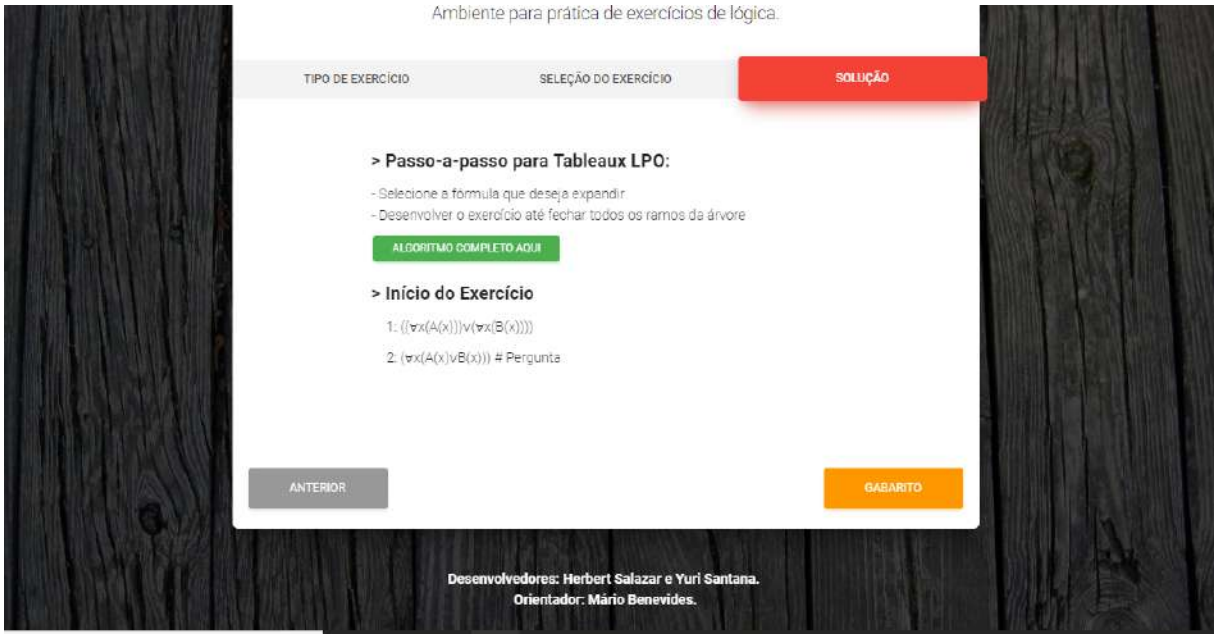


Figura 25 – Tableaux LPO - Exercicio 119

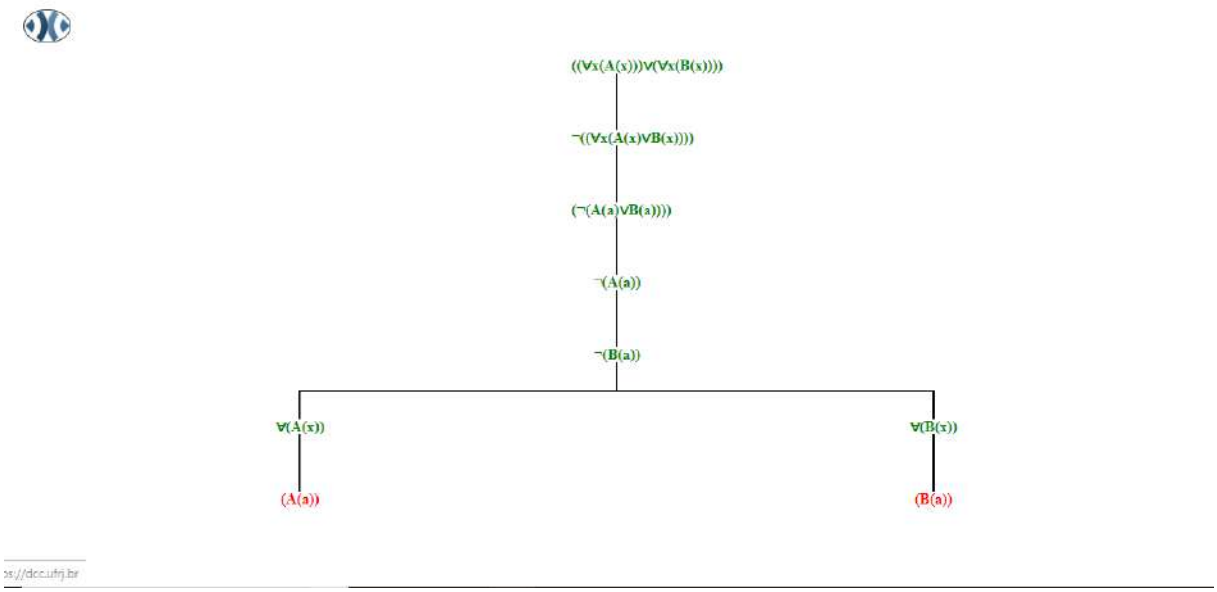


Figura 26 – Tableaux LPO - Função Gabarito Exercicio 119

5.2.2 Semântica

O método da semântica é bastante iterativo, quando o usuário seleciona um exercício é permitido que o usuário escolha insira elementos para compor o seu conjunto Domínio, bem como relações para compor o seu conjuto de relações. O usuário pode colocar pode colocar tantos elementos quanto queira para compor o seu domínio, deste modo o algo-

ritmo expandirá todos os conectivos na fórmula do exercício de acordo com o domínio recebido, assim se o domínio contiver muitos elementos a árvore poderá ter tamanhos colossais. Recomenda-se o uso de domínios de tamanho 2, que são o mínimo necessário para se provar se uma fórmula pode ser validada ou falsificada, porém pode-se testar domínios maiores do que 2 também.

As relações e o domínio podem ser qualquer coisa, porém se as relações não tiverem nenhuma relação com os nós folha gerados eles serão falsos. Segue abaixo imagens que exemplificam o uso da semântica.

Figura 27 – Semântica LPO - Exercício 152

A imagem a seguir contem a árvore do exercício 152, foram usados Domínio=$\{0, 1\}$ e Relações=$\{R(0, 0), R(1, 1)\}$

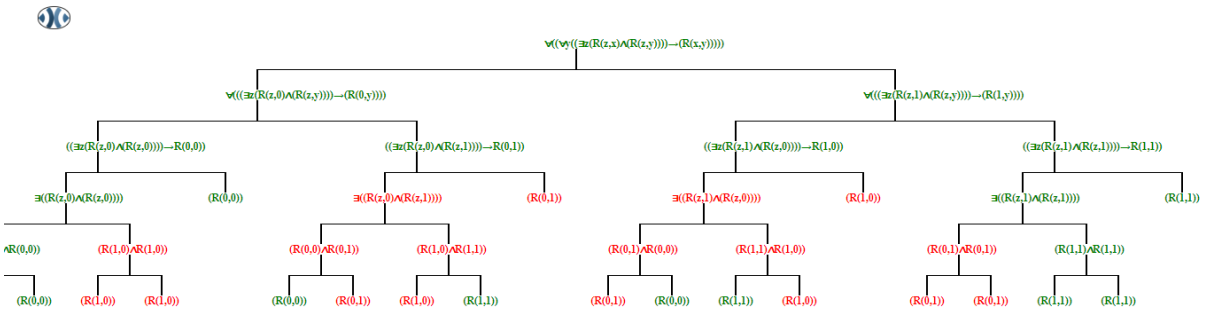


Figura 28 – Semântica LPO - Árvore gerada para o exercício 152

5.3 TRABALHOS FUTUROS

Com este trabalho apresentado tem-se uma grande gama de possibilidades para trabalhos futuros. É possível continuar implementação de features *step-By-Step* para os métodos que não a possuem, bem como outros aprimoramentos similares que expandam a experiência do usuário. Além disto, a proposta inicial deste trabalho também contemplou a possibilidade de permitir a evolução deste mesmo que surjam novas tecnologias no futuro. Fazendo, então, uso da API do projeto é possível construir apenas uma interface gráfica que faça uso de dos métodos já implementados. Em primeira instância pode-se propor a solução de aplicativos Android e IOS para este sistema. E a expectativa do nosso grupo é que o trabalho passe por constante evolução e se torne uma referência para outros trabalhos e projetos.

REFERÊNCIAS

BEDREGAL, B. R. C.; ACIóLY, B. M. **Lógica para a Ciência da Computação**. [S.l.]: Ed. Campus, 2002.

BENEVIDES, M. Apostila de lógica. In: . [S.l.: s.n.], 2008.

BISCHOF, P. V. **Conceitos de interface amigável**. 2012. <http://aprendizdeinterface.blogspot.com/2012/03/conceitos-de-interface-amigavel.html>.

BOOTSTRAP. **Bootstrap**. 2018. <https://getbootstrap.com/>.

BOOTSTRAP. **Introdução**. 2018. <https://getbootstrap.com.br/docs/4.1/getting-started/introduction/>.

BOUWKAMP, K. **The 9 Most In-Demand Programming Languages of 2016**. 2016. <https://www.codingdojo.com//blog/9-most-in-demand-programming-languages-of-2016/>.

CELES, W.; RANGEL, J. L. In: **Apostila de Estruturas de Dados**. [S.l.]: PUC-Rio, 2002. p. 13.9–13.14.

CHAPMAN, S. **Introduction to JavaScript**. 2018. <https://www.thoughtco.com/what-is-javascript-2037921>.

CODERSEYE. **11 Best PHP Frameworks for Modern Web Developers in 2018**. 2018. <https://coderseye.com/best-php-frameworks-for-web-developers/>.

DEVMEDIA. **O que é AJAX?** 2007. <https://www.devmedia.com.br/o-que-e-o-ajax/6702>.

DUMON, O. **How the Internet Changed Science Research and Academic Publishing, Creating the New Research Economy**. 2013. <https://www.huffingtonpost.com/olivier-dumon/how-the-internet-changed_b_2405006.html>.

GROUP, W. B. **Individuals using the Internet (% of population)**. 2018. <https://data.worldbank.org/indicator/IT.NET.USER.ZS>.

GROUP, W. B. **Individuals using the Internet (% of population)**. 2018. <https://data.worldbank.org/indicator/IT.NET.USER.ZS?locations=BR>.

JQUERY. **What is jQuery?** 2018. <https://jquery.com/>.

KAWAKAMI, B. **O que de fato é o Composer?** 2013. <http://www.laravel.com.br/o-que-de-fato-e-o-composer/>.

LARAVEL. **Laravel Documentation**. 2018. <https://laravel.com/docs/4.2/>.

MEDIUM. **Most Used PHP Frameworks in 2017**. 2017. <https://medium.com/@vishva.eleganzit/most-used-php-frameworks-in-2017-73572e562fe9>.

MISIRLAKIS, S. **The 7 Most In-Demand Programming Languages of 2018**. 2018. <https://www.codingdojo.com/blog/7-most-in-demand-programming-languages-of-2018/>.

OCTOVERSE, G. **The fifteen most popular languages on GitHub**. 2018. <https://octoverse.github.com/>.

OPENSOFT. **Web service: o que é, como funciona, para que serve?** 2016. <https://www.opensoft.pt/web-service/>.

PATEL, J. **The 9 Most In-Demand Programming Languages of 2017**. 2017. <https://www.codingdojo.com//blog/9-most-in-demand-programming-languages-of-2017/>.

PAUL; DEITEL, H. **Java: Como Programar. 8 ed.** [S.l.]: Pearson, 2010. 1144 p.

PHP. **O que é o PHP?** 2018. <http://php.net/manual/pt_BR/intro-whatis.php>.

PUTANO, B. **Most Popular and Influential Programming Languages of 2018**. 2018. <https://stackify.com/popular-programming-languages-2018/>.

RIBEIRO, M.; FRANCISCO, R. In: **Web Services REST**. [S.l.]: Instituto Federal Goiano, 2016.

ROBERTO, J. **O que é Laravel? Porque usá-lo?** 2013. <https://medium.com/joaorobertopb/o-que-%C3%A9-laravel-porque-us%C3%A1-lo-955c95d2453d>.

SHANNON, R. **What is HTML?** 2018. <https://www.yourhtmlsource.com/starthere/whatishtml.html>.

SHARP, J. **Visual C# 2010: Passo a Passo**. [S.l.]: Bookman, 2011. 780 p.

SMULLYAN, R. M. **First Order Logic**. [S.l.]: Springer-Verlag, 1968.

SRIDHAR, J. **What Is JavaScript and How Does It Works?** 2017. <https://www.makeuseof.com/tag/what-is-javascript/>.

STACKOVERFLOW. **Most Popular Technologies**. 2018. <https://insights.stackoverflow.com/survey/2018/>.

STATS, I. L. **Total number of Websites**. 2018. <http://www.internetlivestats.com/total-number-of-websites/>.

TECHTERMS. **Framework Definition**. 2013. <https://techterms.com/definition/framework>.

TIOBE. **TIOBE Index for August 2018**. 2018. <https://www.tiobe.com/tiobe-index/>.

TRACTO. **Quantas pessoas têm acesso à internet no mundo?** 2017. <https://www.tracto.com.br/quantas-pessoas-tem-acesso-a-internet-no-mundo/>.

TREINAWEB. **O que é front-end e back-end?** 2017. <https://www.treinaweb.com.br/blog/o-que-e-front-end-e-back-end/>.

TUTORIALSPOINT. **Data Structure and Algorithms - Tree**. 2018. <https://www.tutorialspoint.com/data_structures_algorithms/tree_data_structure.htm>.

W3SCHOOLS. **AJAX Introduction**. 2018. <https://www.w3schools.com/js/js_ajax_intro.asp>.

W3SCHOOLS. **Bootstrap Get Started**. 2018. <https://www.w3schools.com/bootstrap/bootstrap_get_started.asp>.

W3SCHOOLS. **CSS Tutorial**. 2018. <https://www.w3schools.com/css/default.asp>.

W3SCHOOLS. **JavaScript Tutorial**. 2018. <https://www.w3schools.com/js/default.asp>.

W3SCHOOLS. **jQuery Introduction**. 2018. <https://www.w3schools.com/Jquery/jquery_intro.asp>.

W3SCHOOLS. **PHP MySQL Database**. 2018. <https://https://www.w3schools.com/php/php_mysql_intro.asp>.

W3SCHOOLS. **What is HTML?** 2018. <https://www.w3schools.com/html/html_intro.asp>.

APÊNDICES

APÊNDICE A – LISTA DE SERVIÇOS PROVIDOS PELA LOGICAPI

Nome: ListarCategorias
Endereço: /api/exercicios/listarCategorias
Tipo de requisição: GET
Descrição: Lista todas as categorias de exercícios presentes no banco de dados do *Web Service* (tableaux, resolução, tableaux de primeira ordem, etc)
Formato de Entrada: Não é necessária nenhuma entrada para esse serviço
Formato de Saída: JSON

Chave	Tipo	Descrição
id	Inteiro	Id da categoria no banco de dados do Web Service
tipo	String	Nome da categoria

Tabela 1 – Saída do serviço ListarCategorias

Nome: getListas
Endereço: /api/exercicios/getListas
Tipo de requisição: GET
Descrição: Mostra todas as listas de exercícios cadastradas para uma determinada categoria
Formato de Entrada: JSON

Chave	Tipo	Descrição
id	Inteiro	Id da categoria da qual serão recuperadas as listas de exercícios relacionadas

Tabela 2 – Entrada do serviço getListas

Formato de Saída: JSON

Chave	Tipo	Descrição
id	Inteiro	Id da lista de exercícios no banco de dados do Web Service
nome	String	Nome da lista de exercícios
categorias_id	Inteiro	Id da categoria relacionada àquela lista de exercícios

Tabela 3 – Saída do serviço getListas

Nome: getListas
Endereço: /api/exercicios/getListas
Tipo de requisição: GET
Descrição: Mostra todas as listas de exercícios cadastradas, independentemente da categoria
Formato de Entrada: Não é necessária nenhuma entrada para esse serviço

Formato de Saída: JSON

Chave	Tipo	Descrição
id	Inteiro	Id da lista de exercícios no banco de dados do Web Service
nome	String	Nome da lista de exercícios
categorias_id	Inteiro	Id da categoria relacionada àquela lista de exercícios

Tabela 4 – Saída do serviço getListas

Nome: listarExercícios
Endereço: /api/exercicios/listarExercicios
Tipo de requisição: GET
Descrição: Lista todos os exercícios cadastrados de uma categoria (tableaux, resolução, dedução natural, etc.)
Formato de Entrada: JSON

Chave	Tipo	Descrição
categoria_id	Inteiro	Id da categoria da qual serão recuperados os exercícios relacionados

Tabela 5 – Entrada do serviço listarExercícios

Formato de Saída: JSON

Chave	Tipo	Descrição
id	Inteiro	Id do exercício no banco de dados do Web Service
sentenca	String	String contendo as formúlas que compõem o exercício sendo a última fórmula a pergunta de consequência lógica que se deseja verificar

Tabela 6 – Saída do serviço listarExercícios

Nome: listarExercícios
Endereço: /api/exercicios/listarExercicios
Tipo de requisição: GET
Descrição: Lista todos os exercícios cadastrados para uma lista de exercícios específica
Formato de Entrada: JSON

Chave	Tipo	Descrição
lista_id	Inteiro	Id da lista de exercícios da qual serão recuperados os exercícios relacionados

Tabela 7 – Entrada do serviço listarExercícios

Formato de Saída: JSON

Chave	Tipo	Descrição
id	Inteiro	Id do exercício no banco de dados do Web Service
sentenca	String	String contendo as formúlas que compõem o exercício, sendo a última fórmula a pergunta de consequência lógica que se deseja verificar

Tabela 8 – Saída do serviço listarExercícios

Nome: getExercicio
Endereço: /api/exercicios/getExercicio
Tipo de requisição: GET
Descrição: Pega um exercício específico
Formato de Entrada: JSON

Chave	Tipo	Descrição
id	Inteiro	Id do exercício a ser recuperado

Tabela 9 – Entrada do serviço getExercicio

Formato de Saída: JSON

Chave	Tipo	Descrição
-	Array	Array contendo cada fórmula do exercício em uma posição, sendo a última fórmula a pergunta de consequência lógica que se deseja verificar.

Tabela 10 – Saída do serviço getExercicio

Nome: verificaFórmula
Endereço: /api/exercicios/verificaFormula
Tipo de requisição: GET
Descrição: Verifica se uma fórmula está nos padrões do Web Service e retorna a formula com os símbolos lógicos
Formato de Entrada: JSON

Chave	Tipo	Descrição
formulas	String	Fórmula que se deseja verificar (deve-se começar com parênteses)

Tabela 11 – Entrada do serviço verificaFórmula

Formato de Saída: JSON

Chave	Tipo	Descrição
formulas	String	Fórmula convertida de texto para os símbolos lógicos
1	Inteiro	Retorna 1 quando há algum erro na fórmula

Tabela 12 – Saída do serviço verificaFórmula

Nome: resolveExercicioResolucao
Endereço: /api/resolucao/fullSteps
Tipo de requisição: GET
Descrição: Resolve um exercício cadastrado no banco de dados da ferramenta utilizando o método de resolução
Formato de Entrada: JSON

Chave	Tipo	Descrição
exercicio	Inteiro	Id do exercício que se deseja resolver

Tabela 13 – Entrada do serviço resolveExercicioResolucao

Formato de Saída: JSON

Chave	Tipo	Descrição
-	Array	Array contendo a resolução completa do exercício, com cada posição do array sendo um novo passo. Algumas posições do array podem conter também uma string explicando qual o passo foi aplicado a partir daquele momento.

Tabela 14 – Saída do serviço resolveExercicioResolucao

Nome: resolveExercicioResolucao
Endereço: /api/resolucao/fullSteps
Tipo de requisição: GET
Descrição: Resolve um exercício qualquer utilizando o método de resolução
Formato de Entrada: JSON

Chave	Tipo	Descrição
formulas	Array	Array onde cada posição é uma fórmula do exercício, e a última fórmula do array é a pergunta que se deseja responder.

Tabela 15 – Entrada do serviço resolveExercicioResolucao

Formato de Saída: JSON

Chave	Tipo	Descrição
-	Array	Array contendo a resolução completa do exercício, com cada posição do array sendo um novo passo. Algumas posições do array podem conter também uma string explicando qual o passo foi aplicado a partir daquele momento.

Tabela 16 – Saída do serviço resolveExercicioResolucao

Nome: stepByStepResolucao
Endereço: /api/resolucao/stepByStep
Tipo de requisição: GET
Descrição: Resolve um passo de um problema utilizando o método de resolução
Formato de Entrada: JSON

Chave	Tipo	Descrição
operação	String	String da operação a ser realizada. Valores possíveis: "negPergunta", "FNC", "SeparaE", "SeparaOu", "PassarNotParaDentro", "notnot"
qtd_formulasSelecionadas	Inteiro	Quantidade de fórmulas selecionadas pelo usuário
formulas	Array	Fórmulas selecionadas pelo usuário para aplicação da regra escolhida. Cada fórmula deve estar em uma posição do array

Tabela 17 – Entrada do serviço stepByStepResolucao

Formato de Saída: JSON

Chave	Tipo	Descrição
-	Array	Array com o resultado da aplicação das regras de resolução nas fórmulas selecionadas, onde cada posição do array é o resultado de uma das fórmulas

Tabela 18 – Saída do serviço resolveExercicioResolucao

Nome: validaExercício
Endereço: /api/resolucao/validaExercicio
Tipo de requisição: GET
Descrição: Verifica se um exercício inserido pode ser resolvido pela LogicAPI ou se há algum erro de sintaxe
Formato de Entrada: JSON

Chave	Tipo	Descrição
formulas	Array	Array com as fórmulas do exercício, sendo que a última fórmula contém a pergunta que se deseja verificar

Tabela 19 – Entrada do serviço validaExercício

Formato de Saída: JSON

Chave	Tipo	Descrição
-	-	Retorna vazio se o exercício pode ser resolvido pela LogicAPI
1	Inteiro	Retorna 1 se há algum erro e o exercício não pode ser resolvido

Tabela 20 – Saída do serviço resolveExercicioResolucao

Nome: tableaux
Endereço: /api/tableaux
Tipo de requisição: GET
Descrição: Resolve um exercício utilizando o método de tableaux
Formato de Entrada: JSON

Chave	Tipo	Descrição
exercicio	Inteiro	Id do exercício cadastrado no Web Service

Tabela 21 – Entrada do serviço tableaux

Formato de Saída: JSON

Chave	Tipo	Descrição
-	Array	Retorna um array onde a primeira posição contém um string com a estrutura de lista não ordenada do HTML que corresponde à árvore de tableaux e a segunda posição contém um array com informações relevantes sobre os nós da árvore: A fórmula (chave 'info'), a fórmula que gerou aquele nó (chave 'formulaGeradora') e se o nó representa um ramo fechado ou não (chave 'fechado')

Tabela 22 – Saída do serviço tableaux

Nome: tableauxLPO
Endereço: /api/tableauxLPO
Tipo de requisição: GET
Descrição: Resolve um exercício utilizando o método de tableaux de primeira ordem
Formato de Entrada: JSON

Chave	Tipo	Descrição
exercicio	Inteiro	Id do exercício cadastrado no Web Service

Tabela 23 – Entrada do serviço tableauxLPO

Formato de Saída: JSON

Chave	Tipo	Descrição
-	Array	Retorna um array onde a primeira posição contém um string com a estrutura de lista não ordenada do HTML que corresponde à árvore de tableaux de primeira ordem e a segunda posição contém um array com informações relevantes sobre os nós da árvore: A fórmula (chave 'info'), a fórmula que gerou aquele nó (chave 'formulaGeradora') e se o nó representa um ramo fechado ou não (chave 'fechado')

Tabela 24 – Saída do serviço tableauxLPO

Nome: semantica
Endereço: /api/semantica
Tipo de requisição: GET
Descrição: Resolve um exercício utilizando o método de semântica
Formato de Entrada: JSON

Chave	Tipo	Descrição
exercicio	Inteiro	Id do exercício cadastrado no Web Service
dominio	Array	Array contendo o domínio, ou seja, os valores necessários para gerarem a árvore que serão usados para instanciar as funções das fórmulas lógicas.
relacoes	Array	Array contendo as relações, ou seja, funções instanciadas com valores que podem serem assumidos baseados no domínio.

Tabela 25 – Entrada do serviço semantica

Formato de Saída: JSON

Chave	Tipo	Descrição
-	Array	Retorna um array onde a primeira posição contém um string com a estrutura de lista não ordenada do HTML que corresponde à árvore de tableaux de primeira ordem e a segunda posição contém um array com informações relevantes sobre os nós da árvore: A fórmula (chave 'info'), a fórmula que gerou aquele nó (chave 'formulaGeradora') e se o nó representa um ramo fechado ou não (chave 'fechado')

Tabela 26 – Saída do serviço semantica

Nome: novoExercicioDN
Endereço: /api/deducaoNatural/novoExercicio
Tipo de requisição: GET
Descrição: Inicia um exercício de Dedução Natural, dando um id para cada uma das fórmulas
Formato de Entrada: JSON

Chave	Tipo	Descrição
formulas	Array	Fórmulas do exercício, sem a pergunta

Tabela 27 – Entrada do serviço novoExercicioDN

Formato de Saída: JSON

Chave	Tipo	Descrição
-	Array	Retorna um array contendo o id de cada pergunta, o id de contexto, o seu texto, qual ícone será utilizado para a fórmula, indicativo se é uma suposição e um array contendo todas as fórmulas do contexto para cada uma das fórmulas selecionadas. Contexto refere-se ao escopo da dedução natural.

Tabela 28 – Saída do serviço novoExercicioDN

Nome: formataPerguntaDN
Endereço: /api/deducaoNatural/formataPergunta
Tipo de requisição: GET
Descrição: Formata a pergunta da Dedução Natural, convertendo as strings em símbolos lógicos
Formato de Entrada: JSON

Chave	Tipo	Descrição
pergunta	String	Pergunta do exercício

Tabela 29 – Entrada do serviço formataPerguntaDN

Formato de Saída: JSON

Chave	Tipo	Descrição
-	String	Pergunta do exercício formatada com símbolos lógicos

Tabela 30 – Saída do serviço formataPerguntaDN

Nome: stepDN
Endereço: /api/deducaoNatural/step
Tipo de requisição: GET
Descrição: Executa um passo da Dedução Natural, verificando se é possível aplicar a regra selecionada e retornando o resultado da aplicação da regra, caso seja possível
Formato de Entrada: JSON

Chave	Tipo	Descrição
step	String	Regra a ser aplicada. Valores possíveis: supor (nova suposição), elimNot (eliminação do not), elimE (eliminação do e), excImp (exclusão do implica), incE (inclusão do e), abs (declarar absurdo), raa (redução ao absurdo), incImp (inclusão do implica), incNot (inclusão do not), incOu (inclusão do ou), excOu (exclusão do ou), stepOu (iniciar a segunda suposição da exclusão do ou), elimOu (eliminação do ou)
supor	String	Fórmula a ser suposta, no caso de aplicação de regra de suposição
selecionados	Array	Array com fórmulas selecionadas pelo usuário para aplicação das regras
incluir	String	Variável ou expressão lógica a ser incluída quando utilizada a regra da inclusão do Ou
atual	Array	Estado atual da Dedução Natural, contendo todas as fórmulas e seus contextos. Essa árvore é necessária para a aplicação das regras

Tabela 31 – Entrada do serviço stepDN

Formato de Saída: JSON

Chave	Tipo	Descrição
-	Array	Retorna um array contendo o resultado da aplicação da regra, com o seu id, o id de contexto, o seu texto, qual ícone será utilizado para a fórmula, indicativo se é uma suposição e um array contendo todas as fórmulas do contexto do resultado. Contexto refere-se ao escopo da dedução natural.
erro	String	No caso em que uma regra não possa ser aplicada, é retornado uma string com uma mensagem informando o motivo da não aplicação da regra

Tabela 32 – Saída do serviço stepDN